Student Name: Arun Chakkyadath Chandran **Student ID: S230543676**

Project Supervisor: Dr. Simon Stuttaford

NEWCASTLE UNIVERSITY

School of Engineering

Electrical and Electronic Engineering

General Guidance for Final Report for EEE8097 (Individual Project)

When preparing the Technical Report, authors should consider the following points:

1) It is recommended that the overall length of the final report should be no more than 8000 words. The overall length of the report should be commensurate with the importance, or appropriate to the complexity, of the work.
2) Final reports must provide sufficient information to allow readers to perform similar experiments or calculations and use the reported results. The report must contain useable and fully described information.
3) Do not pad the report with trivial information just to boost the word count of the report unnecessarily. Focus on quality.
4) Authors must convince the reader (and importantly the EEE8097 assessors) of the scientific and technical merit of the report.
5) Avoid information, which is peripheral to, or outside, the main area technical area of interest. For example, analysis of the semiconductor physics of a photovoltaic cell need not be discussed if the main purpose of the report is to analyse total harmonic distortion in grid connected inverters for renewable energy applications.
6) Results must be supported by adequate data and critical details.
7) Results of importance should be discussed, and critically evaluated.
8) Additional validation or explanation is required when extraordinary or unexpected results are reported.
9) IMPORTANT: All work should be your own. Ensure you write each section of the report using your own words. All final reports will be checked for potential plagiarism via TurnItIn. Students are strongly advised to familiarize themselves with the University guidance on plagiarism before commencing work on the final report. As a minimum, students should check out the *Understanding Plagiarism* video on the library website: Plagiarism - Managing Information - LibGuides at Newcastle University (ncl.ac.uk)
10) Not knowing the University Guidelines on plagiarism will not be accepted as an excuse in the event of any plagiarism follow up.

Student Declaration

I, Arun Chakkyadath Chandran confirm that this report and the work presented in it are my own achievement.

I have read and understand the penalties associated with plagiarism.

Signed:  Date: 23/08/2024

Propose Optimal Conditions for EMG Pattern Recognition Systems by Comparing and Analysing with Various Conditions, Features, and Classifiers.

Author: Arun Chakkyadath Chandran (S230543676)
Supervisor: Dr. Simon Stuttaford

Abstract

Electromyographic (EMG) data processing may result in the usage of an artificial prosthetic arm. The pattern recognition-based myoelectric control system produces promising results in prosthetic limbs, but the problem is to develop optimal solutions in terms of signal quality, processing time, feature selection, and classifier models. This project uses the publicly published initial version of the Ninapro database to deliver optimal solutions in several challenging sectors of electromyographic-based hand pattern categorisation systems. According to the investigation, the primary challenges were selecting the right window size, filtering condition, relevant feature, and classifier. To identify the most optimal answer for each difficulty, the project is divided into four sub-sections: general findings, combined dataset analysis, individual subject analysis, and hybrid model concepts. In the section on general results, give the best solution for the selection window size and filtering specifications. The ideal window size is 500-250, and the best filtering condition occurs when the stimulus and restimulus values, as well as repetition and rerepetition, are equal. The problem of selecting relevant features is solved by using the statistical analysis approach and counting outliers. As an outcome, the relevant features are slope sign changes, wavelength, total power, and time-frequency energy. Furthermore, for further analysis of the statistical non-parametric Friedman test in individual subjects, the time-frequency energy feature outperforms the other features. Several experiments are used to select the best classifier, including comparing accuracy and processing time, consistency test, calculating classification error, and determining true positive values in target labels. Based on experimentation, the best categorisation models are k-Nearest Neighbour and random forest. Finally, according to the study, the solution to these issues is to create a hybrid model. It gives a comprehensive solution to all the difficulties highlighted.

Keywords: Electromyography, Windowsize, Filtering conditions, Relevant feature, Best classifier, Hybrid model.

Table of Contents

Introduction

The most effective approach for controlling an artificial prosthetic hand is to identify hand gestures or patterns using electromyography data. However, this technology has more potential than standard mechanically restricted movement (open-close), which frequently reduces naturalness and efficiency. Furthermore, as a real-world application for people, it is critical in human-computer interaction, virtual reality, robotics, and upper/lower hand amputee assistive devices. The basic operation relies on skeletal muscle activation, which interferes with electrical information from EMG (electromyography) sensors. Recent research has shown that it is feasible to extract hand gesture patterns with greater accuracy from various muscle actions [1,2]. Using the publicly accessible NinaPro dataset, this study investigates solutions for problems with window size, filter condition, optimal features, and classifier selection [3, 4]. This Ninapro dataset comprises sEMG data from 27 healthy people doing 12 hand movement tasks and one rest position. The 1 to 8 channels contain data from eight sEMG electrodes evenly distributed around the forearm at the height of the radio humeral joint. Channels 9 and 10 provide signals from the flexor and extensor digitorum muscles. Stimulus is a pattern that repeats a movement, whereas restimulus repeats the same pattern but for a longer period. The terms repetition and rerepetition relate to the repeating of both the stimulus and the restimulus [3].

The objective is to investigate methods for enhancing classification of hand patterns using the Ninapro DB1 (Experiment 1) dataset. First, learn about the fundamental components of pattern classification systems. Then, examine each component in depth to see what difficulties are currently occurring. Finally, offer the most optimised solution for the specific challenges using conditional comparisons and statistical tests, and then suggest a better hybrid model. According to the extensive examination, the complete system includes data preparation, windowing, filtering, feature extraction, and classification into appropriate class labels or hand patterns. Objectively, it is divided into four sections (A, B, C, and D), which include general findings, combined dataset analysis, individual subject analysis, and hybrid model proposals. For further examination, each section is broken into multiple experiments.

- Part A: General Findings.

    1. Determine the optimal window size.
    2. Identify the Best Filtering Condition:

- Part B: Combined Dataset Analysis (27 Subjects).

    1. Find the best features using statistical analysis.
    2. Analyse Outliers in Best Features.
    3. Find the best classification model based on a variety of experiments.

- Part C: Individual Subject Analysis.

    1. Discover the best features for each subject.
    2. Find the best classification model for each subject.
    3. Perform a statistical analysis test on three different features.

- Part D: Propose the Hybrid Model.

In the general results section (Part A), first do an experiment to establish the best window size based on the trade-off between accuracy and processing time for various classifier models. This experiment clearly demonstrates how window size influences model accuracy and processing time. Second, determine the optimal filtering conditions by computing the two parameters, mean squared error (MSE) and signal-to-noise ratio (SNR), under various stimulus, restimulus, repetition, and rerepetition circumstances. Calculate the accuracy of various classifier models with corresponding criteria at the same time. This comparison will offer a clear

direction for an optimal filter condition, utilising the trade-off between mean squared error, signal-to-noise ratio, and classifier accuracy. It utilises MSE and SNR measures to optimise the trade-off between noise reduction and signal quality [5].

In Part B, I integrate 27 separate datasets into a single dataset and apply statistical analysis methods (mean, median, and standard deviation) to classify them as the best, moderate, and least informative features. Second, identify outliers in the best features; this helps to understand how data samples are organised in each feature. Finally, choose the optimal classifier model using several conditional tests [6, 7] (accuracy and processing time, mean accuracy and standard deviation, classification error, and confusion matrix).

In Part C, I first perform individual dataset analysis (experimenting on each of the 27 patients) to discover the most significant features in each. Second, identify the optimal classification model (KNN or Random Forest) for every subject. Finally, use the Friedman statistical non-parametric test to identify the most significant feature. This test ranks features based on their testing accuracy across multiple classifiers [8].

Part D proposes a hybrid model based on the optimisation findings from every part, as well as a feature combination-based hybrid model that incorporates features from many domains. The integration of time, frequency, and time-frequency domains strengthens machine learning models and improves their performance [9]. The MATLAB tool handled data preprocessing, feature extraction, and extraction time calculations, while the Python software handled hand pattern classification using multiple machine learning models and classification time calculations.

Theoretical Background and System Description

1. System Architecture

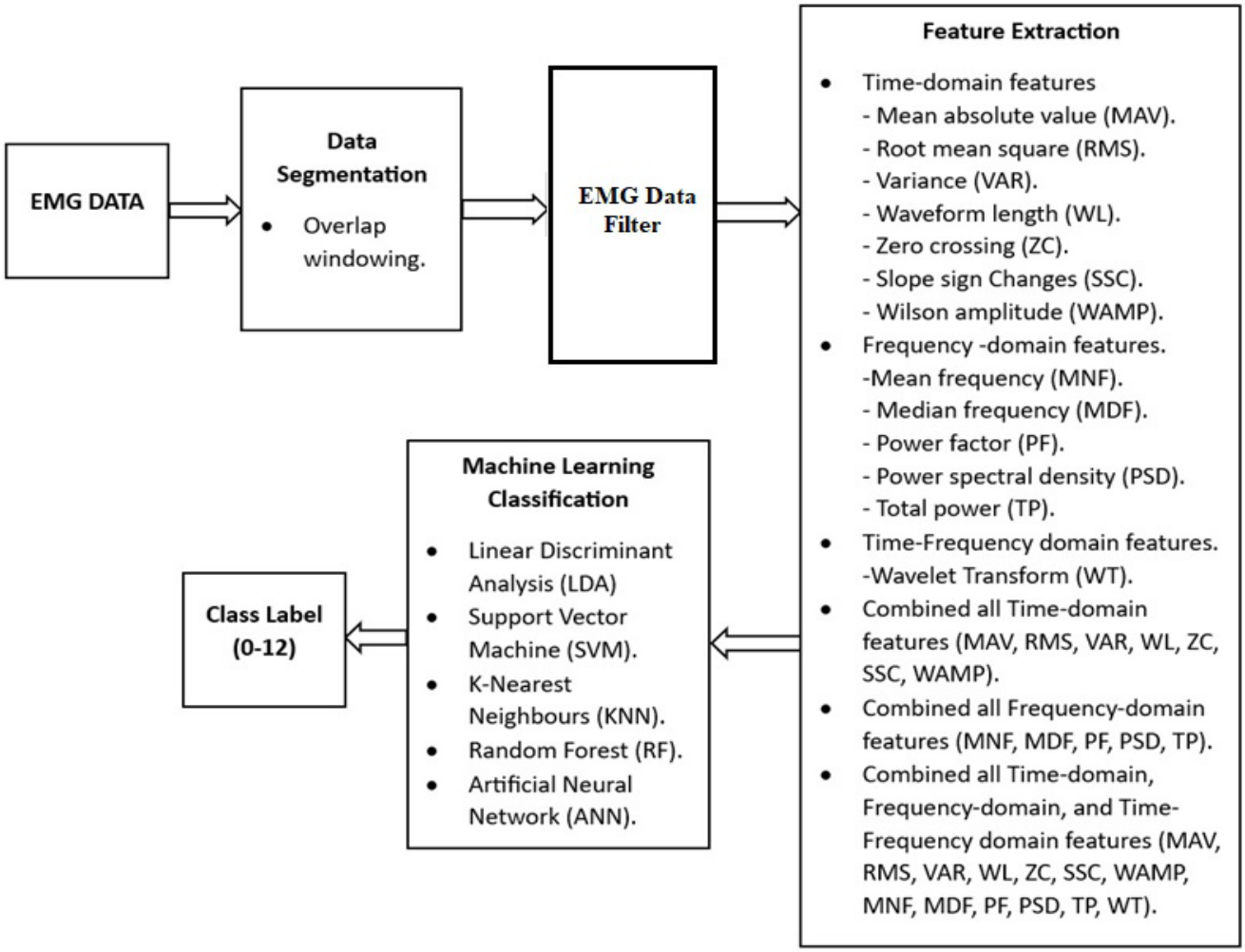


Figure 1. System architecture for pattern recognition using electromyography [10,11,12]

The system architecture processes real-time muscle activity signals to extract relevant features for prosthetic device control and pattern recognition. The data preprocessing stage extracts 10 channel EMG data from each subject, combining them in raw-wise order without changing class labels. The windowing process divides the extracted data into multiple segments, using the sliding window method to capture transitional information. The filter condition is applied to the current window, extracting features like time, frequency, and time-frequency domains. Machine learning classification models (LDA, SVM, KNN, Random Forest, and ANN) are trained to predict the accuracy of testing and training. The 13 target labels are used for predicting unseen data. Signal processing techniques like windowing, filtering, feature extraction, and pattern classification ensure the reliability and validity of surface electromyography (sEMG) data analysis, which is crucial for real-time prosthetic control, rehabilitation, and ergonomic studies.

2. EMG Data

The Ninapro dataset includes ten EMG channels in DB1, with the sEMG electrode detecting voltage changes in each channel to indicate muscle fibre contraction activity. These forms of muscle activity investigations aid in the analysis of muscle coordination, activation patterns, muscle fatigue, and motor control. These muscle signals offer the most significant contribution to pattern recognition. The flexor digitorum muscles flex the fingers, whereas the extensor digitorum muscles lengthen them. Figure 2 depicts the various EMG signals for ten channels.

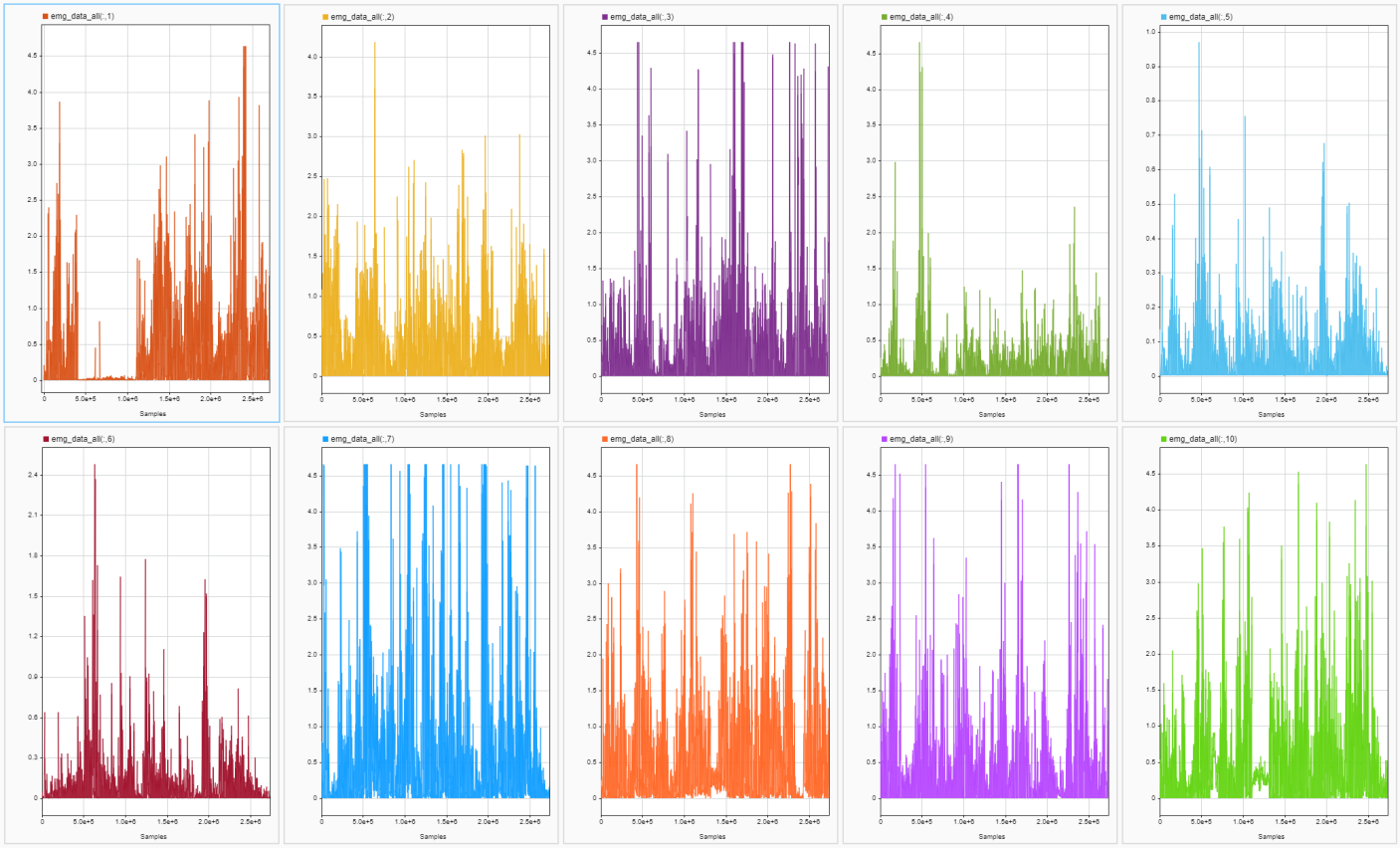


Figure 2. EMG signals for 10-channels.

2.1 Stimulus, Repetition, Restimulus, and Rerepetition.

This study uses various techniques to analyze hand movements. Stimulus is used to ensure consistent hand movements and data collection. Repetition is the same visual prompt or action after initial exposure, examining its consistency and dependability over time. Repeating the same instructions ensures accuracy and reliability. Performance is assessed by assessing how successfully a person executes actions over time. Pattern identification aids in analyzing signal variations during the same action. Rerepetition ensures consistent results across multiple trials and monitors performance changes as the number of studies increases. Figure 3 illustrates these signal representations.

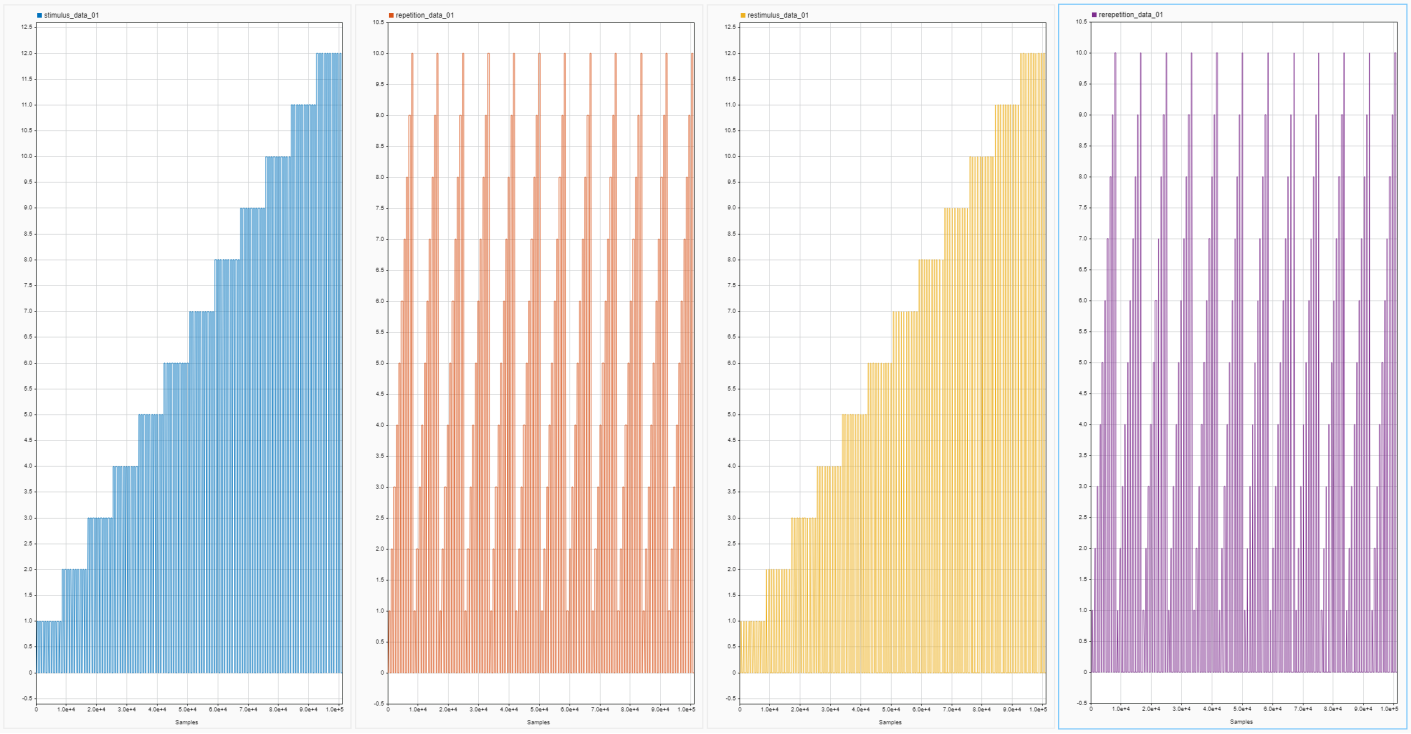


Figure 3. Signal representation of stimulus, repetition restimulus, and rerepetition.

There are noticeable points; the width of the restimulus and rerepetition signal are reduced as compared with their respective stimulus and repetition signals (see figure 4). The reason for reducing the signal width is due to multiple reasons, such as neural adaptation (the neural response of restimulus and rerepetition is reduced over time) and cognitive habituation (with repeated trials, the human cognitive system is more adept at processing restimulus and repetition actions). See figure 4. Graphical differences between stimulus and restimulus, as well as between repetition and rerepetition.

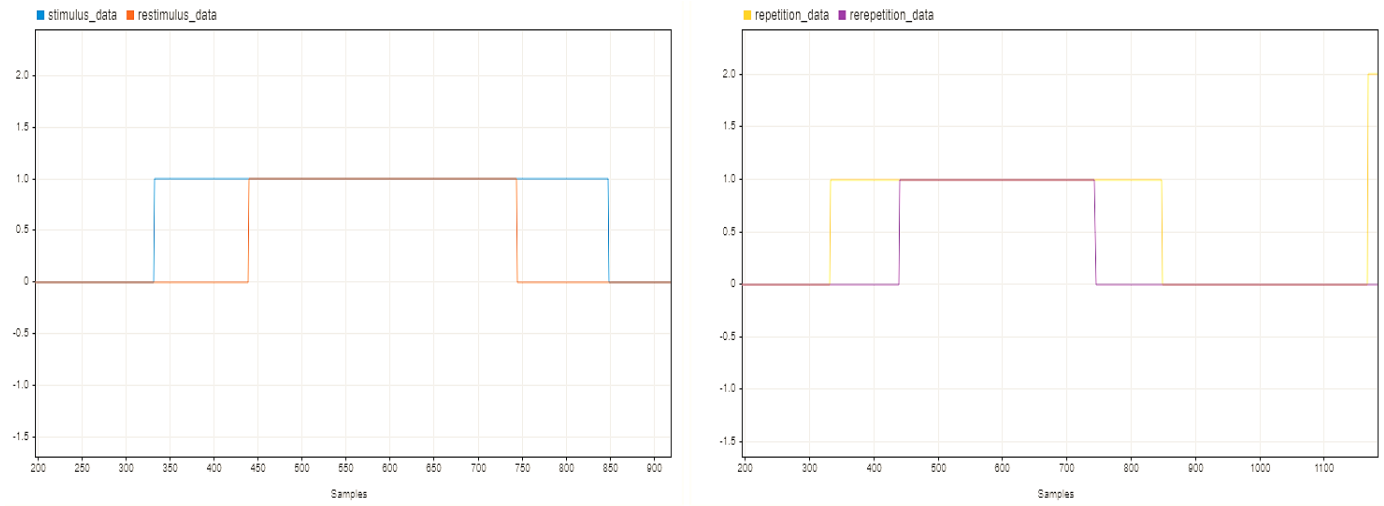


Figure 4. Graphical differences between stimulus and restimulus, as well as between repetition and rerepetition.

3. Windowing Process

The sliding window strategy is chosen in this project since the EMG data is a time series that varies constantly throughout time. The sliding window separates the EMG data into certain window lengths, such as overlapping or non-overlapping segments (window). Each window's right features are retrieved and saved in a defined array. In this project, certain parameter names are utilised, such as winsize (actual window size), wininc (increase window size), numwin (number of windows), and curwin (current window).

The actual window sizes are 800, 500, 250, 150, 100, and 50, and increment window sizes are 400, 250, 75, 50, and 25. Figure 5 illustrate the respective window sizes, and their increment window sizes with retrieved samples. The window size specifies how many data samples are included. For example, if winsize = 500 is meant by each window, it will contain 500 samples from the signal or data. Window increment size (sliding window) is specified as how much window moves forward for the next segment. For example, if wininc = 250 is meant by each sliding or overlapping window, it will start 250 samples after the previous window's start. The number of windows (number) in each class (0 to 12) is calculated using the below equation (1). Where N is the total number of samples with the respective class. The '+1' ensures the initial window at the start of the signal; without including it, it misses the first segment window.

Finally, the representation of window size in seconds depends on the sampling rate of the EMG electrode; here, the Ninapro DB1 experiment is using 10 Otto Bock MyoBock 13E200 electrodes [13]. For example, consider sampling frequency is 1000 Hz and winsize is 500. Using equation (2), the answer is 0.5 second or 500 milliseconds.

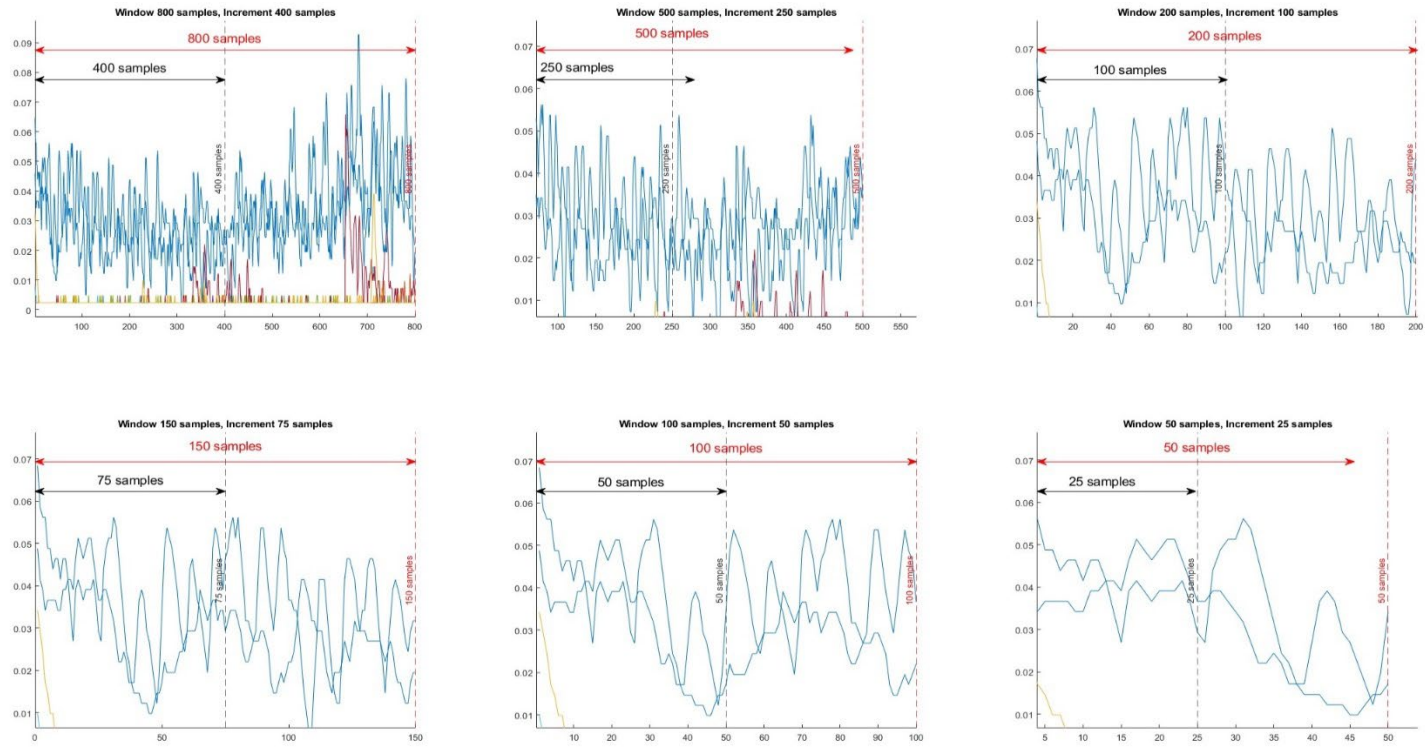


Figure 5. Window size for 800-400, 500-250, 200-100, 150-75, 100-50, 50-25.

$$numwin = \left[\frac{N - winzise}{wininc}\right] + 1 \qquad (1)$$

Equation (1). Calculation for number of windows

$$T = \frac{winsize}{Fs} \qquad (2)$$

Equation (2). Calculation for window size time in seconds.

4. Filtering

EMG data filtering relies on stimulus, restimulus, repetition, and rerepetition values, but their arrangement can be inconsistent. For instance, a combination of stimulus = 0 and restimulus = 3 indicates preparation state, causing annotation delays. The system is labeled rest state, but an upcoming action or pattern is already marked in the restimulus. To ensure consistency, accurate labeling, and noise reduction, the filtering should be chosen where both the stimulus and restimulus values are equal. This method is also applicable to repetition and rerepetition.

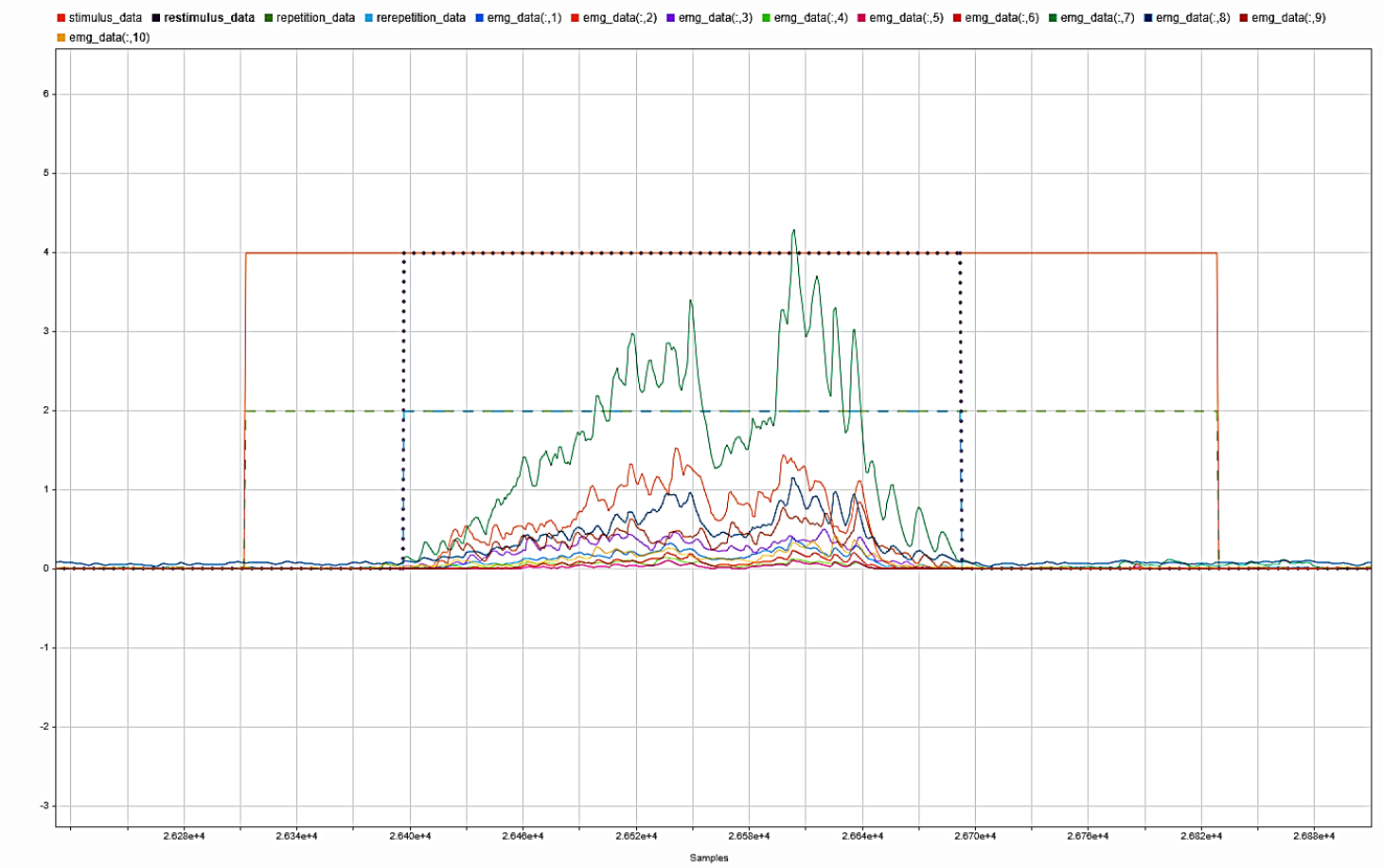


Figure 6. EMG signal extraction when stimulus and restimulus are equal.

5. Feature extraction

Various features are extracted from time, frequency, and time-frequency domains after the data filtering. Seven time-domain features are extracted, such as mean absolute value, to provide the average of the absolute values of the EMG signal [14]. Root mean square provides an indication of the energy content and power of the EMG signal [15]. Variances indicate the power of the EMG signal [16]. Waveform length provides the cumulative length of the EMG waveform over time [14]. Slope Sign Changes The number of times the slope of the EMG signal changes sign, indicating the frequency of muscle activity changes [14]. Zero crossing is the number of times the EMG signal crosses the zero line, indicating the frequency of the signal's oscillations [14]. Willison Amplitude is the count of the number of times the amplitude difference between consecutive samples exceeds a threshold, related to muscle contraction levels [17].

In the frequency-domain feature, mean frequency (MNF) is the average frequency of the signal, providing muscle fatigue and the frequency content of the signal [18]. Median Frequency (MDF), the frequency that divides the power spectrum into two equal parts, is often used to assess muscle fatigue [18]. Peak frequency (PF), the frequency at which the power spectrum of the EMG signal is at its maximum [18]. Power spectral density (PSD), a measure of the power distribution of the signal across different frequencies, is used for analysing frequency components [19]. Total Power (TP) is the total energy of the EMG signal over a period, representing the overall power of the signal [20, 23]. Time-frequency energy (TF-Energy) features from the time-frequency domain and provide the total energy of the EMG signal over a period, representing the overall power of the signal [21, 22]. Finally, combine seven time-domain features (COMB-TD), five frequency-domain features (COMB-FD), seven time-domain and five frequency-domain features (COMB-TD & FD), and combine all time, frequency, and tim-frequency domain features (COMB-TD & TFD.

6. Classification Model and Class labels

After feature extraction from various domains, these data fed to the different machine learning classification models (LDA, SVM, KNN, RF, and ANN). The target labels of all these classification models range from 0 to 12 (as illustrated in Figure 7), and each class label is represented by a distinct hand pattern.

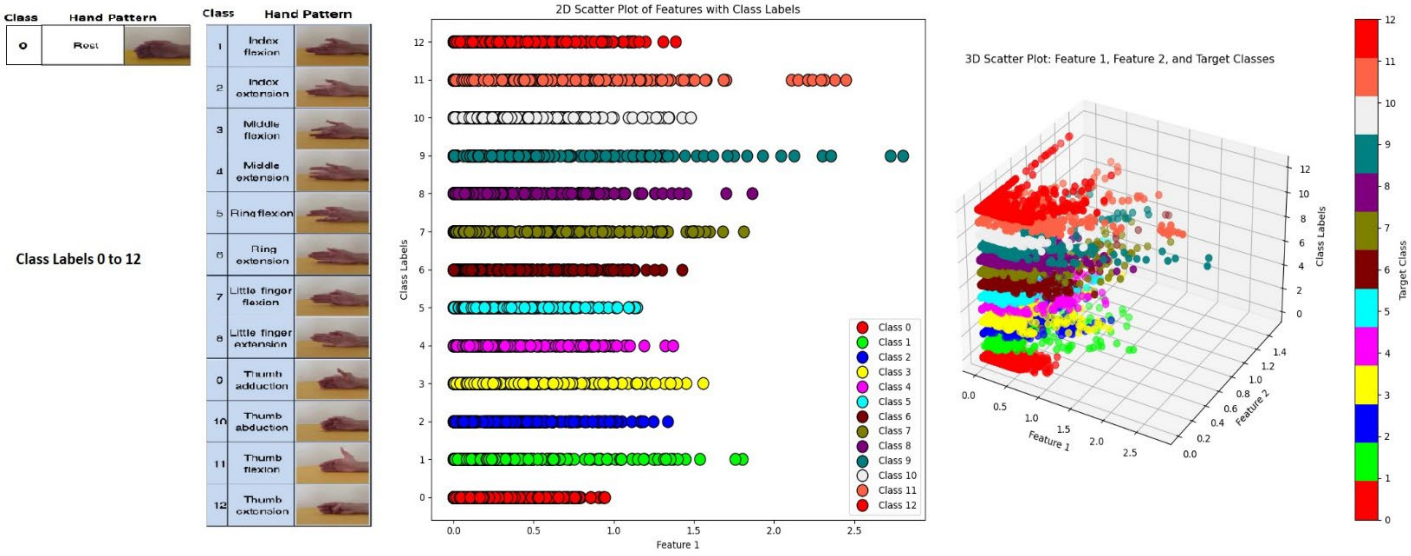


Figure. 7 Classification label [3]

Literature Review

In this review of the literature, I would like to address the key findings about electromyography systems, as well as the challenges brought up by several research investigations and suggestions for advancement.The part of overall processing of EMG signals includes, a more sophisticated method of windowing, advanced filtering techniques, methods for feature extraction, and combining of multi-model data. Finally, discuss the significant machine learning model that has shown superior performance in Ninapro datasets.

Preprocessing is the first step of pattern recognition techniques for reducing the interferences of the signals, such as signal acquisition noise, electromagnetic interference, and instability of the signal due to the wrong placement of electrodes [24]. To address these filtering issues, the top filtering models in EMG signal processing are wavelet transform filtering, Kalman filtering, and Notch filtering. Wavelet Transform Filtering (WTF) is one of the best methods to avoid the EMG noise reduction. In D. Farina and R. Merletti et al. [25], address the various noise factors are affected by sEMG electrodes such as movement artifacts, electromagnetic interference, crosstalk, internal noise, and electrocardiographic (ECG) artifacts are discussed. The solution to improve the system performance mainly depend on their hardware components and algorithms. Foe improving design architecture, size, higher signal-to-noise ratio these parameters sufficiently reduce the noise contents [26,27,28]. These techniques are mitigating the impact of inherent noise. In the case of movement artefacts to distort the EMG signals, it's mainly due to the displacement of electrodes and cables [29]. The solution to the concern is to use proper conductive gel to help maintain a stable interface, use wavelet transform filter techniques to reduce the motion artefact noises, and use proper electrode placement [30, 31]. Electromagnetic noise due to external devices; the solution to this problem is using adaptive filtering techniques such as FIR and IIR notch filters. These filters are effectively removing the electromagnetic interference noise [32]. Laguerre filters are the reliable technique for removing the power-line interference (PLI) [33]. The common in signal interpretation due to the size of the electrode and placement of muscle region [35]. To use of high-pass spatial filters is much capable to avoid the subcutaneous tissue thickness, muscle fatigue, and electrocardiographic artefacts issues [36,37,38]. The proposed solutions effectively address the various noise issues in EMG signals and lead to more accurate and reliable EMG signal interpretation in clinical diagnostics, rehabilitation devices, and human-machine interfaces.

The most common method used in electromyography field or time-series data is the sliding window method, and it's powerful for analysing the dynamic muscle activity. The key factors in windowing are the selection of the actual window and sliding window sizes. Window size is a trade-off between many factors in electromyography systems, especially accuracy and processing time. A larger window size can capture the general pattern of the signal but might lead to underfitting by missing short-term significant events. Smaller window sizes are providing higher temporal resolution and able to capture the small changes, so it probably leads to overfitting [39, 40]. The characteristics of the EMG signal are dependent on the muscle activity and their movement. Therefore, the fixed windowing method is not applicable for all types of movements. The solution to the issue is to use the adaptive windowing technology so it automatically changes the window size as per the characteristics of the EMG signals [41]. However, the cross-validation method is a considerable option to analyse the consistency of the window process [42]. One of the real-time limitations is that smaller window sizes capture a high amount of data, and the cost of computation is high. The suitable solution of the computational overhead to implement the dimensionality reduction techniques such as linear discriminant analysis (LDA) without loss of information [43]. The best trade-off between actual window size and incremental window size is fifty percent, and it balances the computation time and quality of the signals [44].

The time-domain, frequency-domain, and time-frequency domain features are commonly extracted from the Ninapro dataset. The time-domain features contain the characteristics of statistical and waveform-based information of the EMG signal. In the frequency domain, using Fourier transform, the features are derived by transforming the signal into the frequency domain. The time-frequency domain features are a combination of both and provide more detailed information about the non-stationary EMG signals. In-time-domain features are widely used due to their simplicity and low computational power. In the context of the latest research on pattern recognition systems, deep learning methods are used for getting higher accuracy and robustness. However, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) provide higher accuracy as compared to traditional features [45, 46]. The main challenges are noticed variability of EMG signals, data collection, and normalization. The variability of electromyography (EMG) signals is measured by the accuracy of the model being changed due to the changes of individuals, which affect the consistency. Secondly, for ensuring the normalisation technique is consistent for feature extraction with an extensive amount of data. The solution to these challenges is to standardise the EMG signals across different users and sessions, such as z-score normalisation or min-max scaling to reduce variability. In the context of an EMG signal, the wavelet transform (WT) feature is one of the powerful features for analysing non-stationary EMG signals. It contains both time and frequency domain information, and this combined information nature will provide higher accuracy in particular machine learning models [47]. The main challenges are the complexity of the feature extraction and ensuring that feature is computationally efficient. Finally, the entropy-based feature extraction method plays critical roles if the electromyography signals are of an irregular nature. Phinyomark et al. introduced how the integration of entropy-based both time and frequency domain features significantly increases the accuracy and stability of pattern recognition systems [48]. The main challenges are to find the best features and avoid the redundant or irrelevant features because it reduces the system performance. The solution for this issue is to implement the feature selection method such as recursive feature elimination and LASSO (Least Absolute Shrinkage and Selection Operator). Moreover, the hybrid method or integration of features from different domains provides a comprehensive solution to the above issues [49].

LDA is a linear classifier, and it works better in a linear dataset, and that best separates the classes (hand pattern) in the dataset. If the data is a non-linear combination, LDA is less efficient in terms of accuracy prediction and least computation time with a low memory requirement [1]. As discussed in various research papers, the solution to solving the non-linear data is to use higher complex algorithms such as support vector machines (SVMs) and neural networks (CNNs and RNNs) that can handle the nonlinear relationships in the data [50, 51]. The use of kernel functions for the SVM algorithm is more efficient in non-linear complex data due to the hyperplane mechanism [52]. The most common challenges for the data preprocessing part are features selection, nonlinearity in data, training time complexity, and class imbalance. The solution for this issue to choose relevant features with help of feature engineering method and reducing the dimension of the data. Due to these techniques to avoid the overfitting and improve the computational efficiency. However, the method of regularisation and resampling techniques is better to reduce the complexity of the data, training time, and provide better class imbalance issues.

k-Nearest Neighbours (k-NN) is more efficient in the case where the data distribution is unknown or of a nonlinear nature. Due to the simplicity, the computation time is comparatively low and performs well in moderate-sized datasets but might be overfitting [53]. Overall, as per the research paper analysis, the common challenges are computational time, scalability, distance metric choice, and choice of k value. The issues of computational complexity and scalability can solve nearest neighbours (ANN) algorithms (e.g., KD trees, ball trees). For improving scalability, the clustering method is highly efficient. Random Forest is a method of

ensemble learning, and it includes multiple decision trees during training and provides the result based on the mode of individual trees. As per the research paper analysis, it outperformed in complex data and indicates the accuracy range of 85 to 95 percent. The noticeable point is that the number of trees is higher, and the computation cost is also high. These issues can be solved by using a threshold value that is determined due to the experiment between the quantity of characteristics and the accuracy of the model [54].

The artificial neural network (ANN) model is more powerful as compared with the above classifiers, but the common factor is the higher computation time. As a result, the complex algorithms are not suitable for real-time applications due to the key factor processing time. The highlighted characteristic is automatically learning the data pattern with multiple epochs and providing better classification accuracy. One of the drawbacks for ANNs is that it requires a large amount of data, so obviously it led to higher computation power. The challenges are overfitting, high dimensionality, and hyperparameter tuning. The implied solutions for finding the overfit are to experiment with the performance accuracy in training and testing datasets. It helps to mitigate the overfitting by validating the model's performance on unseen data. The high-dimensionality issue was solved by applying a diffusion map for dimensionality reduction in fuzzy classification. Finally, the hyperparameter tuning issue was solved by experimenting with different hidden layers and activation functions to find the optimal configuration [55].

Simulation Details

This project simulation is divided into two parts: the first (Part 1) uses MATLAB for feature extraction and extraction time calculation, and the second is a machine learning classification model created in Python using a Jupyter Notebook. The reason for separating is that the MATLAB tool offers superior capabilities for mathematical calculation in the case of feature extraction, however, Python provides a diverse set of machine learning libraries to do numerous tasks in comparison to the MATLAB tool. For a better method, utilise multiple tools depending on their performance in different contexts, and then combine both files to use batch files (Windows) or shell scripts (Linux) via a sequence of instructions run by the command prompt. It enables you to automate many processes by executing several commands in order.

The simulation operates in two separate scenarios. The first step is to integrate the 27 individuals from the Ninapro DB1 dataset into a single data set. Second, work with individual participants from the Ninapro DB1 dataset. This type of experiment was used to study and compare the assessment, processing time, and feature selection in both domains. However, both techniques might serve to create a more sophisticated hybrid model. The final hybrid model is trained using all the participants' data; thus, it has various properties and a wider prediction range than individual trained models. Figure 8 shows the flowchart for the total simulation process.

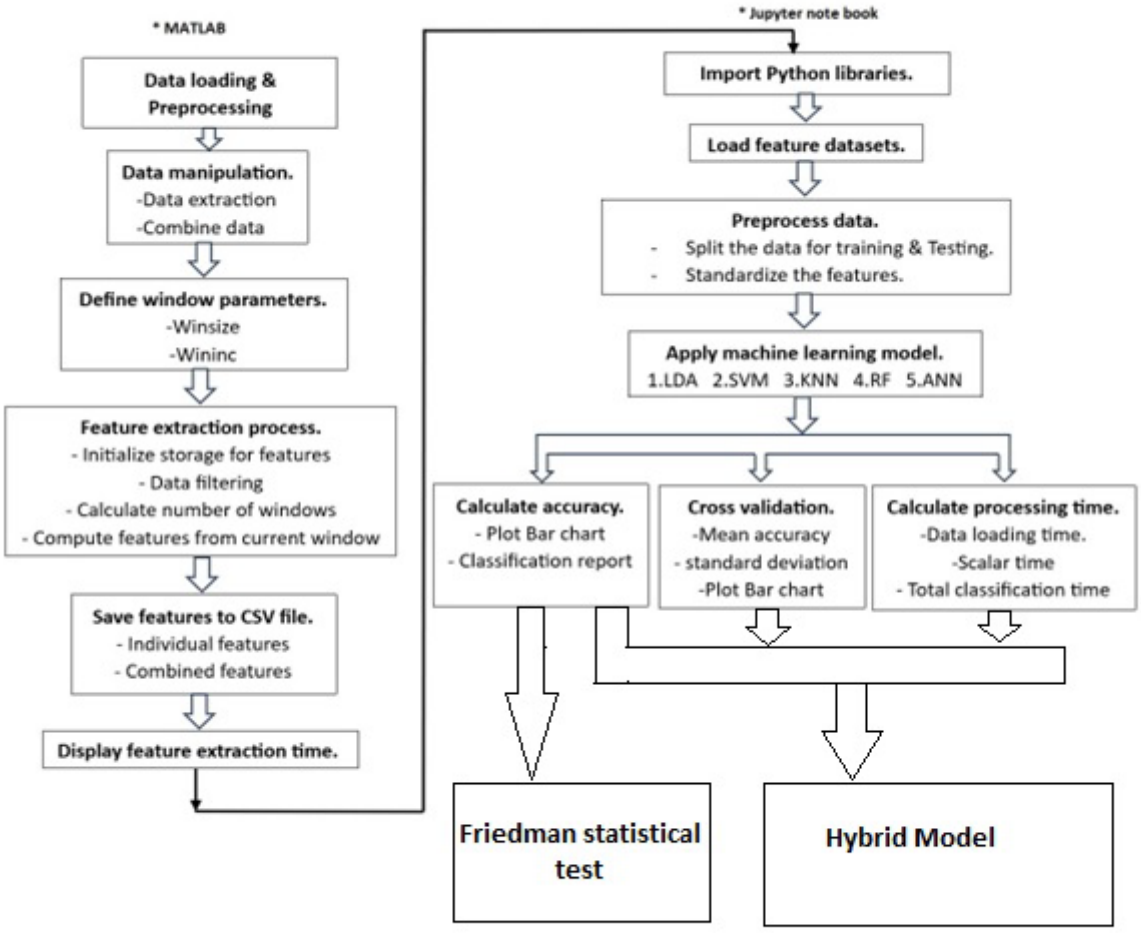


Figure. 8 Flowchart for overall simulation.

Part:1 (MATLAB)

1. Data Loading and Preprocessing

The data loading and preprocessing part starts by recording the overall execution time using the command "startTime = datetime('now');". The "clc" command can clear the MATLAB command window. Using the "load" function, 27 participants or subject data loaded and stored in separate variables (data_01, data_02, etc.). After loading 27 participants, the script prints the size of various data matrices, including EMG data, stimulus, restimulus, repetition, and rerepetition. This way of data structure representation ensures the data is correctly loaded and provides a quick overview. See figure 9 for data loading and preprocessing.

```
Subject 01:
  EMG data size: [101014 10]
  Stimulus data size: [101014 1]
  Restimulus data size: [101014 1]
  Repetition data size: [101014 1]
  Rerepetition data size: [101014 1]
Subject 02:
  EMG data size: [100686 10]
  Stimulus data size: [100686 1]
  Restimulus data size: [100686 1]
  Repetition data size: [100686 1]
  Rerepetition data size: [100686 1]
```

Figure.9 Data loading and preprocessing.

2. Data Manipulation

The section of data manipulation first does specific variable extraction (EMG, stimulus, restimulus, repetition, rerepetition) from loaded data for each participant using the "eval" function. This function is used to evaluate the string for loaded 27 participant data and assign respective variable names for each data, for example, data_01 to emg_data_01, stimulus_data_01, restimulus_data_01, repition_data_01, and rerepition_data_01, similarly repeated up to 27 participant data. The command "sprint" creates a variable name from the respective subjects (data_01, data_02 to data_27). Secondly, data combination: the individual variables data for all participants are combined into a single dataset using concatenation ([]). For example, emg_data_all, stimulus_data_all, restimulus_data_all, repetition_data_all, and rerepitition_data_all. Finally, display the combined data size with their variable data size also. See figure 10 for the data manipulation process.

```
Combined data size of 1 to 27 Participiants: [2731393 14]
  EMG data_all size: [2731393 10]
  Stimulus_data_all size: [2731393 1]
  Restimulus_data_all size: [2731393 1]
  Repetition_data_all size: [2731393 1]
  Rerepetition_data_all size: [2731393 1]
```

Figure. 10 Data Manipulation process.

3. Define window parameters.

Windowing is a crucial step in electromyography (EMG) for analyzing non-stationary signals. It divides the continuous signal into small segments and analyzes them independently, allowing for the extraction of features over time. Window size (winsize) refers to the number of data points within each segment, while window increment (wininc) is the number of samples moved forward for the next segment. This overlapping method ensures no significant information is lost between segments and improves feature extraction smoothness. The selection of winsize and wininc is a trade-off between temporal resolution and computational efficiency. See figure 11: Window parameter initialisation.

```
% Define window size and increment
winsize = 500;  % Window size in samples (800, 500, 200, 150, 100, 50)
wininc = 250;   % Window increment in samples (400, 250, 100, 75, 50, 25)
```

Figure.11 Window parameter Initialisation.

4. Feature extraction process

4.1 Initialise storage for features.

Initialise the array storage for each feature across all classes. These include time-domain, frequency-domain, and time-frequency domain. Similarly, initialise the timing variables to record the time taken for extracting each feature type across all classes. See figure 12 for feature storage initialisation.

```
% Initialize storage for each feature across all classes
all_mav_features = [];
all_rms_features = [];
all_var_features = [];
all_wl_features = [];
all_zc_features = [];
all_ssc_features = [];
all_wamp_features = [];

all_mnf_features = [];
all_mdf_features = [];
all_pf_features = [];
all_psd_features = [];
all_tp_features = [];

all_tf_energy_features = [];
```

Figure. 12 feature storage initialisations

4.2 Data filtering and display number of windows in each class

The EMG data is filtered based on the condition that both the stimulus and restimulus values match the current class label, and the value of repetition matches the value of rerepetition. Finally, this process loops through different classes, filters the data by stimulus and repetition labels, calculates the number of windows in each calss, and allocates storage for various features. See Figure 13 for data filtering and number of windows.

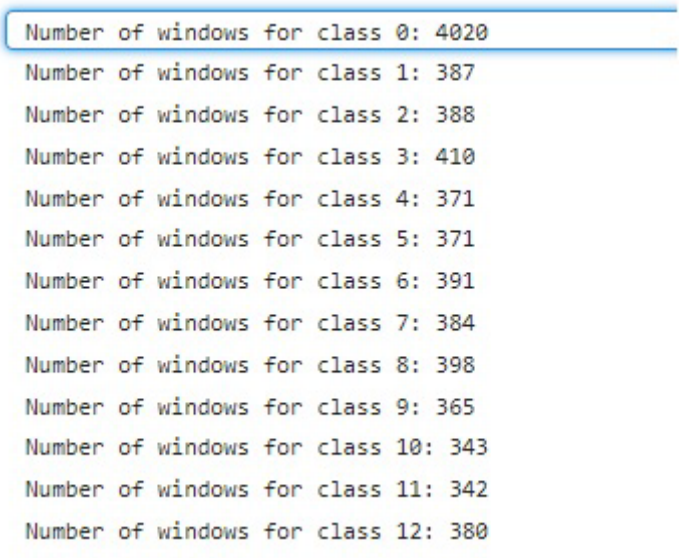


Figure. 13 data filtering and number of windows.

4.3 Compute features from current window and extraction time calculation.

The feature extraction and their computing time are recorded for each window to evaluate the efficiency and performance of the process. The extracted features have different aspects of the signal, including amplitude, frequency content, and energy distribution. These features are stored in pre-allocated matrices for each feature type. After extracting these features for each class, append the class label to the last column for each feature matrix. See figure 14 for feature extraction and their computation time.

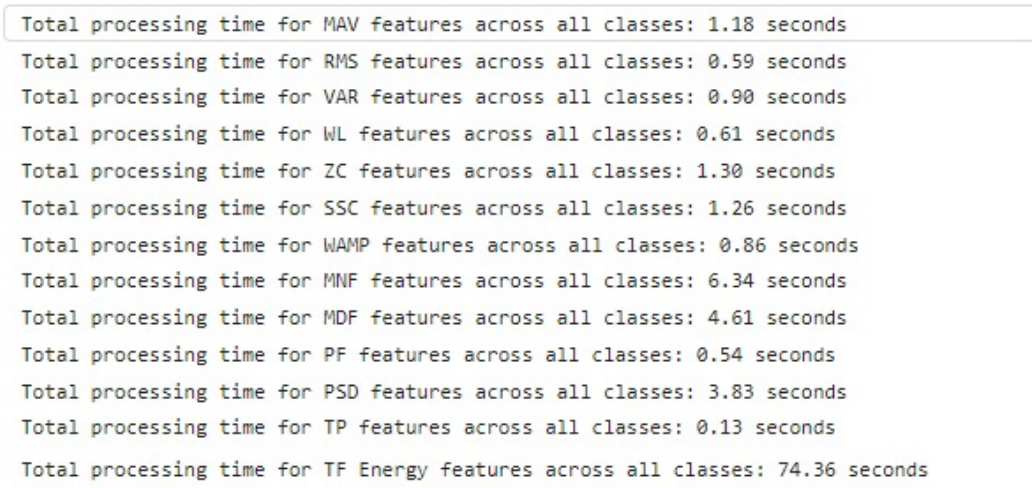


Figure.14 feature extraction and their computation time

5. **Save features to csv file.**

The process is storing features from EMG data to CSV files, then integrating time-domain, frequency-domain, and time-frequency domain features into a single dataset with respective names (for example, combined time and frequency domain feature). Each channel has a dynamic header name, which includes (channel names and target label) Feature 1, Feature 2, and class. Figure 15 illustrates feature saving as csv files.

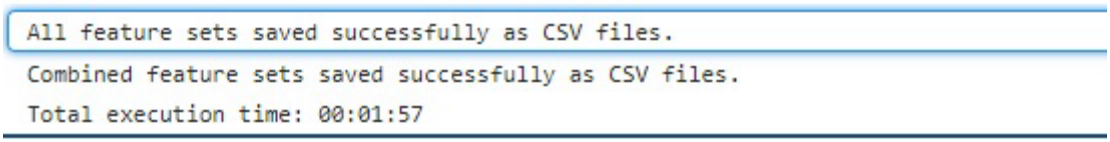


Figure. 15 features saving as csv files.

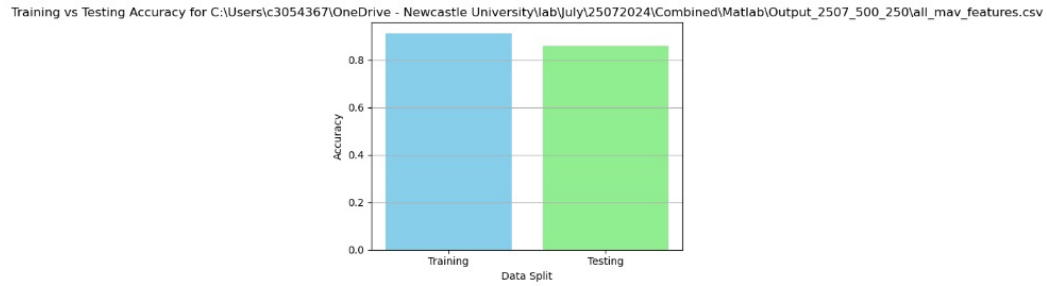
Part: 2 (Python-Jupyter Notebook)

1. Development for classification model.

In part 2, it's mainly about machine learning classification-based models and accuracy, mean accuracy, standard deviation, and classification time. The main libraries are scikit-learn, NumPy, Pandas, Matplotlib, and Seaborn. The Scikit-learn is widely used in machine-learning libraries for data preprocessing, model training, and evaluation. It includes sklearn.discriminant_analysis.LinearDiscriminantAnalysis (LDA), sklearn.svm.SVC (SVM-Classification), sklearn.neighbors.KNeighborsClassifier (KNN classification), sklearn.ensemble.RandomForestClassifier (Random-Forest calcification), and sklearn.neural_network.MLPClassifier (ANN-classification). Secondly, the NumPy library is used for all models for numerical operation and handling arrays. Pandas' library is used for data manipulation and preparation. For data visualisation and plotting, Matplotlib and Seaborn are used.

In the case of accuracy, calculation is evaluated by various machine learning models such as linear discriminant analysis (LDA), support vector machines (SVM), K-nearest neighbours (KNN), random forests (RF), and artificial neural networks (ANN). Each classifier follows a consistent workflow: first load the extracted feature dataset and split it into features (channels or columns in a dataset) and target labels (class labels). Then, after data split into training and testing sets, features were scaled using "StandardScaler" to standardise the input. Each classifier is trained on scaled training data, and predictions are made on the test set. The performance of the model is based on their testing and training accuracies. The results are visualised through bar plots with the help of the "matplotlib" library. Additionally, a detailed classification report, including precision, recall, and F1-score, is generated for better comparison between the classifier models. See figure 16 results for accuracy and classification report.

```
Accuracy for testing on C:\Users\c3054367\OneDrive - Newcastle University\lab\July\25072024\Combined\Matlab\Output_2507_500_250
\all_mav_features.csv: 85.96%
Accuracy for training on C:\Users\c3054367\OneDrive - Newcastle University\lab\July\25072024\Combined\Matlab\Output_2507_500_25
0\all_mav_features.csv: 91.28%
```



Classification Report:

	precision	recall	f1-score	support
0	0.89	0.97	0.93	1212
1	0.87	0.73	0.79	124
2	0.85	0.88	0.86	120
3	0.70	0.79	0.74	106
4	0.88	0.88	0.88	113
5	0.92	0.88	0.90	108
6	0.86	0.80	0.83	116
7	0.84	0.85	0.84	124
8	0.81	0.81	0.81	114
9	0.78	0.62	0.69	112
10	0.70	0.54	0.61	103
11	0.84	0.73	0.78	91
12	0.85	0.57	0.69	122
accuracy			0.86	2565
macro avg	0.83	0.77	0.80	2565
weighted avg	0.86	0.86	0.85	2565

Figure. 16 results for accuracy and classification report.

Secondly, for calculating mean accuracy and standard deviation for each model with the help of cross-validation techniques. This technique ensures the stability or robust assessment of model performance. For each classifier (LDA, SVM, KNN, RF, and ANN), the dataset is divided into multiple folds, and the model is trained and tested on different subsets of the data. Calculate the accuracy scores in each fold and compute the mean accuracy; it indicates the average performance of the model across different subsets of the data. The standard deviation is also calculated to assess the variability or consistency of the model's performance. The library “random_state” ensures that the initialisation of the model parameters is consistent across multiple runs, which helps in stabilising results. Overall, this approach provides information on how each model performs on average and how stable the performance is across different data splits. See figure 17 result for mean accuracy and standard deviation.

```
Results for MAV_Features:
Mean Test Accuracy: 60.36%
Standard Deviation of Test Accuracy: 1.49%

Results for RMS_Features:
Mean Test Accuracy: 61.19%
Standard Deviation of Test Accuracy: 1.82%

Results for VAR_Features:
Mean Test Accuracy: 60.75%
Standard Deviation of Test Accuracy: 1.44%

Results for WL_Features:
Mean Test Accuracy: 59.36%
Standard Deviation of Test Accuracy: 1.73%
```

Figure. 17 Result for mean accuracy and standard deviation.

For calculating the classification time for each feature set with respective models based on required time for both training and prediction phases. This is achieved by calculating the start and end times of the classification processes for each feature set and model combination. This result provides a clear understanding of the computational efficiency of each feature with respective models and valuable insights into their practical performance and suitability for real-time applications. However, performing statistical analysis on each feature using mean, median, and standard deviation. It's helping to understand data distribution. Standard deviation indicates the amount of variation or dispersion from the mean. Secondly, calculate the outliers of those features. The outlier indicates the sum of data distribution, central tendency, variability, and presence of noise. See figure 18. for statistical analysis tests.

```
Summary statistics for C:\Users\c3054367\OneDrive - Newcastle University\lab\July\25072024\Combined\Matlab\Output_2507_500_250
\all_ssc_features.csv:
                mean  median         std  range
Feature1   175.376959   177.0   54.655574  385.0
Feature2   138.166667   157.0   86.713669  393.0
Feature3   148.563275   157.0   68.556909  368.0
Feature4   117.756491   109.0  105.348250  402.0
Feature5   134.362222   113.0   89.456761  376.0
Feature6   147.025263   167.0  103.626058  387.0
Feature7   174.867251   179.0   56.814476  388.0
Feature8   191.514269   184.0   38.743244  346.0
Feature9   132.775673   144.0   95.277312  399.0
Feature10  177.297895   184.0   62.752823  354.0
```

Figure 18 Statistical analysis test

Finally, calculate the confusion matrix for the best model and provide a detailed breakdown of the model's performance by showing the number of true positives, true negatives, false positives, and false negatives. Additionally, in the individual data set, perform the accuracy comparison and find the best features and subjects out of 27. At last, perform the Friedman non-parametric test in the best features (wavelength, total power and time -frequency energy) of individual subjects. See figure 19. for Friedman test statistic and P-value

```
Processing wl_features for S27 from file: C:\Users\c3054367\OneDrive - Newcastle University\lab\Individual\Output_files\S27_wl_
features.csv
Accuracy for testing on C:\Users\c3054367\OneDrive - Newcastle University\lab\Individual\Output_files\S27_wl_features.csv: 81.7
1%
Processing tp_features for S27 from file: C:\Users\c3054367\OneDrive - Newcastle University\lab\Individual\Output_files\S27_tp_
features.csv
Accuracy for testing on C:\Users\c3054367\OneDrive - Newcastle University\lab\Individual\Output_files\S27_tp_features.csv: 87.4
0%
Processing tf_energy_features for S27 from file: C:\Users\c3054367\OneDrive - Newcastle University\lab\Individual\Output_files
\S27_tf_energy_features.csv
Accuracy for testing on C:\Users\c3054367\OneDrive - Newcastle University\lab\Individual\Output_files\S27_tf_energy_features.cs
v: 83.33%
Friedman test statistic: 31.980582524271846, p-value: 1.1363307019687559e-07
There is a significant difference between the feature types.
Results saved to C:\Users\c3054367\OneDrive - Newcastle University\lab\Individual\Output_CSV_file\combined_results.csv
```

Figure.19 Friedman test statistic and P-value.

Results and Discussion

Part A. General Findings

1. Determine the optimal window size.

Finally, considering trade-off aspects, the window size 500-250 (winsize = 500, wininc = 250) is chosen over 800-400 (winsize = 800, wininc = 400). Table 01 compares the average accuracy and processing time for various windows. Figure 20 depicts a graphic representation of window size, accuracy, and processing time. When the window size is raised, the accuracy of all models improves, and processing times drop systematically. Shorter window sizes need more time than larger ones. The requirements include large window sizes (800-400) capable of catching long-term trends and overall signal patterns, but they may result in underfitting by missing short-term important events or temporal resolution of the signal. Smaller window sizes, on the other hand, provide increased temporal resolution while still recording rapid signal changes. However, the resulting obtained features more sensitive to noise and volatility over time. Choosing the appropriate window size involves balancing the model's accuracy against its processing time. Finally, I chose the window size 500-250, which had an average total accuracy of 69.12 percent and an average total processing time of 15.76 seconds. It delivers reasonably greater precision while reducing processing power and short-term fluctuations or noise. The study's goal is to strike a compromise between window size and system processing time to improve classification accuracy while minimising latency. Iterating over subjects and extracting sequential features are among the challenges. For better performance, a parallel windowing solution is advised.

Window size	Parameter	LDA	SVM	KNN	RF	ANN	Avg. Total
800-400	AVG. Accuracy	60.51	63.19	73.69	77.90	72.59	69.58
	AVG. Processing time	12.28	12.72	12.32	12.94	28.54	15.76
500-250	AVG. Accuracy	57.80	60.88	75.50	78.46	72.94	69.12
	AVG. Processing time	14.72	16.99	14.85	18.26	36.49	20.26
200-100	AVG. Accuracy	56.52	58.48	71.94	75.38	70.00	66.46
	AVG. Processing time	18.20	35.17	18.64	27.83	71.15	34.20
150-75	AVG. Accuracy	55.10	56.69	67.14	73.39	68.04	64.07
	AVG. Processing time	20.30	53.60	20.59	33.02	77.39	40.98
100-50	AVG. Accuracy	53.90	55.08	68.32	72.25	66.18	63.15
	AVG. Processing time	20.81	109.55	21.53	169.36	113.05	86.86
50-25	AVG. Accuracy	52.77	53.64	66.61	71.24	64.30	61.71
	AVG. Processing time	31.93	589.42	34.42	71.93	188.09	183.16

Table. 01 Comparison between average accuracy and processing time for different windows

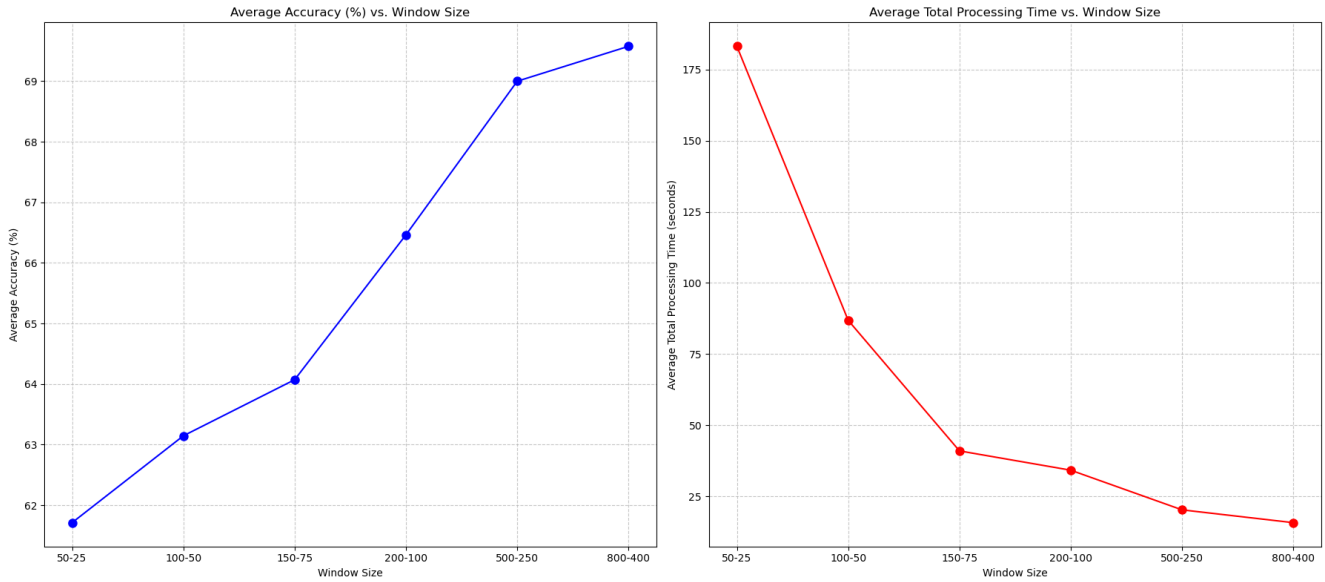


Figure. 20 Graphical representation of window size and accuracy, and window size and processing time

2. Identify the best filtering condition.

Model 4 condition improves SNR moderately after filtering (1.027378 to 1.267982) and has less impact on MSE compared to models 1 and 2 condition. The moderate improvement in SNR after filtering indicates better quality of features used for classification and maintains a relatively low MSE, suggesting good accuracy in predictions.

In terms of accuracy, the filtering condition of model 1 performs very poorly (average accuracy = 62.48%) compared with other models (2, 3, and 4). In Model 2 and Model 3, there is no significant difference in testing, training, or F1score. **Random Forest** and **KNN** generally outperform the other classifiers in terms of accuracy and F1 scores across all models. Finally, **model 4** shows the strongest results overall in terms of testing (75.31%), training (79.85%), and F1-score (0.73). Table 02 and Figure 21 provide a clear comparison incorporating accuracy, F1-score, mean squared error (MSE), and signal-to-noise ratio (SNR) for each model condition.

Filter Conditions	Average MSE (Original)	Average MSE (Filtered)	Average SNR (Original)	Average SNR (Filtered)	Classifiers	Testing Accuracy	Training Accuracy	Weighted Avg F1-Score
Model 1 (Stimulus filtered by Repetition condition 1& 2)	0.123535	0.200242	1.027378 dB	1.413006 dB	LDA	35.08%	36.54%	0.340
					KNN	85.25%	90.32%	0.850
					SVM	40.25%	40.89%	0.390
					RF	89.33%	100.00%	0.890
					Average	**62.48%**	**66.94%**	**0.618**
Model 2 (Stimulus filtered by Repetition condition 5,6,7,8,9,10)	0.123535	0.194971	1.027378 dB	1.361528 dB	LDA	53.44%	53.61%	0.480
					KNN	86.68%	92.13%	0.860
					SVM	56.13%	56.72%	0.510
					RF	91.11%	100.00%	0.910
					Average	**71.84%**	**75.62%**	**0.690**
Model 3 (Stimulus filtered by all Repetition condition (1to 10 and including 0))	0.123535	0.123535	1.027378 dB	1.027378 dB	LDA	53.44%	53.61%	0.480
					KNN	86.68%	92.13%	0.870
					SVM	56.13%	56.72%	0.510
					RF	91.11%	100.00%	0.910
					Average	**71.84%**	**75.62%**	**0.693**
Model 4 (Stimulus = Restimulus filtered by Repetition value = Rerepition)	0.123535	0.168031	1.027378 dB	1.267682 dB	LDA	60.70%	62.07%	0.560
					KNN	88.15%	92.25%	0.880
					SVM	62.96%	65.06%	0.590
					RF	89.43%	100.00%	0.890
					Average	**75.31%**	**79.85%**	**0.730**

Table. 02 Comparison for different filter conditions

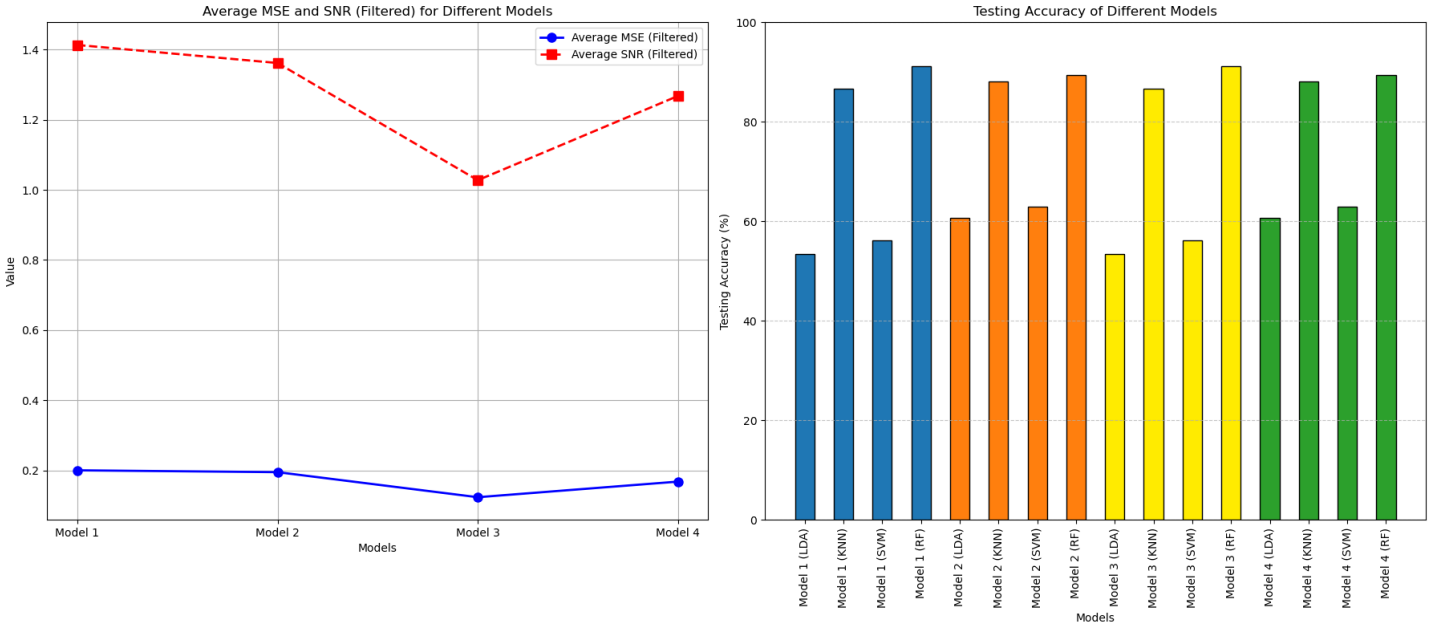


Figure. 21 Graphical representation of different filter condition.

Part B. Combined Dataset Analysis (27 Subjects).

1. Find the best features using statistical analysis.

The root mean square feature (RMS feature) has slightly higher mean and median values (0.20 and 0.10) compared to MAV features (0.17 and 0.08). It indicated RMS features more informative data than MAV features, but higher variability could be challenging. Wavelength (WL Feature) features have high mean (4.71) and median (2.93) values along with a significant standard deviation (5.35), so this feature can capture the complex patterns and distinctions between classes. Slope Sign Change feature (SSC feature) and finally, in time domain sessions, slope sign change, wavelength, and root mean square features provide better accuracy based on their ranges of values, such as mean, median, and standard deviation.

In the frequency domain, the best feature is total power (TP feature) due to the higher mean (.83), median (0.46), and standard deviation (1.07) compared with frequency domain features. Relatively small values mean median and standard deviation indicate that feature capture subtle frequency characteristics. In the time-frequency domain, the time-frequency energy features (TF Energy Features) provide extremely high values for both mean (389.57) and standard deviation (485.15). It indicates that this feature can capture more information but may also be very noisy. Evaluation based mean, median, and standard deviation, each feature is categorised into best, moderate, and less informative features. See table 04 for categorised statistical features.

Dataset	Mean (Average of 10 channels)	Median (Average of 10 channels)	Standard Deviation (Average of 10 channels)
MAV Feature	0.17	0.08	0.24
RMS Feature	0.20	0.10	0.27
VAR Feature	0.04	0.00	0.13
WL Feature	4.71	2.93	5.35
ZC Feature	0.01	0.00	0.11
SSC Feature	153.77	157.10	76.19
WAMP Feature	4.38	0.20	13.91
MNF Feature	0.09	0.08	0.04
MDF Feature	0.05	0.03	0.05
PF Feature	0.0004	0.00	0.0010
PSD Feature	0.0017	0.0009	0.0021
TP Feature	0.83	0.46	1.07
TF_Energy_Feature	389.57	223.09	485.15

Table. 03 Statistical comparison of features

Category	Features
Best informative Features	SSC, WL, TP, TF_Energy
Moderate informative Features	WAMP, RMS, MNF, MDF, PSD
Less informative features	MAV, VAR, ZC, PF

Table. 04. Categorised statistical features

2. Analyse Outliers in best features.

The below box plot (figure. 22) indicates the graphical representation of data distribution, central tendency, variability, and presence of outliers. It's helpful to identify the overall range data distribution in each channel and potential anomalies. The highest number of outliers are presented in total power feature (TP feature) and time-frequency energy feature (TF-Energy feature), indicating high variability. Wavelength feature (WL features), generally a high number of outliers are presented across all channels. In the slope sign change feature (SSC feature), some channels (4, 5, and 6) having no outlier are presented, which might imply more stability in those features. See table 05. Outlier in each channel for different features.

Channels	SSC Features	WL Features	TP Features	TF Energy Features
Outlier channel 1	676	295	335	318
Outlier channel 2	234	286	325	247
Outlier channel 3	252	357	461	445
Outlier channel 4	0	548	614	552
Outlier channel 5	0	594	730	640
Outlier channel 6	0	611	750	661
Outlier channel 7	1,144	545	683	683
Outlier channel 8	375	306	479	416
Outlier channel 9	1	616	638	587
Outlier channel 10	967	497	532	447

Table. 05 Outlier in each channel for different features.

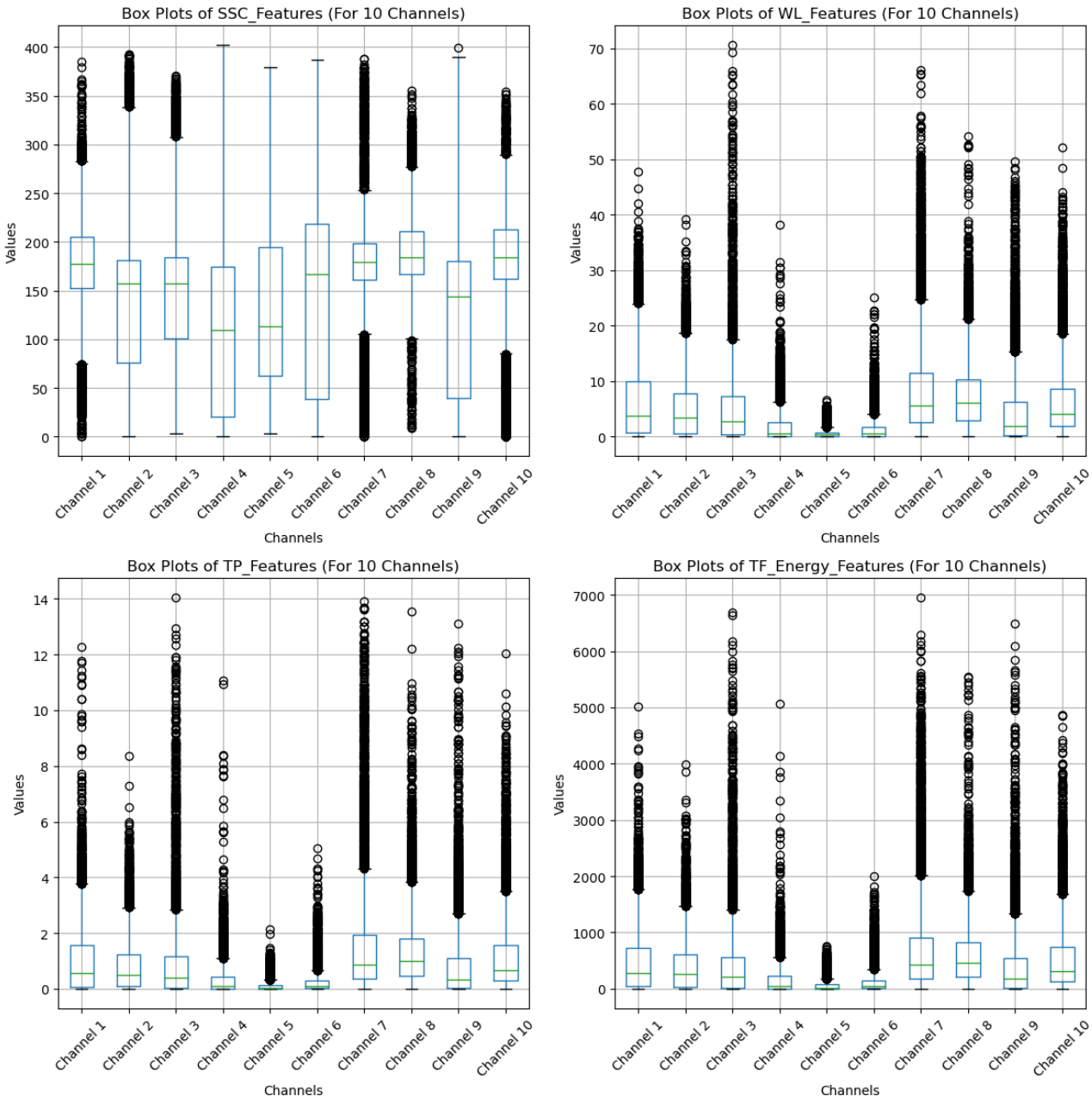


Figure. 22 Graphical representation of data distribution

3. Find the best classification model based on a variety of experiments.

a. Finding the best ML model and features based on accuracy and processing time.

In terms of accuracy comparison, the Random-Forest model has the highest average accuracy (78.46%) across all models, but cost of computation is higher. It performs well in combined all time-domain features (91.7%) and combined all time, frequency, time-frequency domain features (90.21%). Secondly, the KNN model is the second highest performer in average accuracy (75.50%). It also shows strong performance with combined all time-domain features (91.07%) and combined all time, frequency, time-frequency domain features (86.59%). Thirdly, artificial neural networks (ANN) perform quite well, with an average accuracy of 72.94%, and they also perform well in combined all-time domain features (90.49%) and combined all-time, frequency, time-frequency domain features (88.42%). Support Vector Machine (SVM) has an average accuracy of 60.88%, and it performs well in combined all-time, frequency, and time-frequency domain features (80.43%). Finally, Linear Discriminant Analysis (LDA) has the lowest average accuracy (57.80%) and moderate performance in combined all-time, frequency, and time-frequency domain features (70.06%).

In terms of processing time, LDA has the best model, and the average processing time is 14.72 seconds. KNN and SVM have nearly similar processing times of 14.85 seconds and 16.99 seconds, respectively. Random forest has slightly slower average processing time of 18.26 seconds. Finally, ANN is the slowest model, with an average processing time of 36.49 seconds. See table 6.

Window Size: 500-250										
Models	**LDA**		**SVM**		**KNN**		**Random Forest**		**ANN**	
Features	**Total Processing time(seconds)**	**Accuracy**	**Total Processing time(seconds)**	**Accuracy**	**Total Processing time(seconds)**	**Accuracy**	**Total Processing time(seconds)**	**Accuracy**	**Total Processing time(seconds)**	**Accuracy**
MAV	1.1	59.84	3.04	61.72	1.29	85.96	4.38	89.12	34.83	81.52
RMS	0.63	60.7	2.39	62.96	0.85	88.15	4.01	89.43	34.49	83.12
VAR	0.87	59.96	2.57	63	1.1	76.49	4.16	89.86	20.42	72.44
WL	0.66	58.79	2.79	61.25	0.89	87.64	3.97	88.3	7.4	82.61
ZC	1.32	47.25	3.47	47.25	1.53	47.25	1.68	47.29	44.79	47.25
SSC	1.23	44.6	3.98	47.25	1.46	72.24	3.27	70.06	14.17	63.59
WAMP	0.91	54.97	3.22	55.91	1.14	60.66	1.86	68.73	43.78	63.82
MNF	5.61	53.02	7.88	53.65	5.84	60.7	9.48	62.18	26.06	58.13
MDF	4.62	54.42	6.38	55.83	4.85	64.13	8.37	67.25	26.55	60.19
PF	0.62	48.73	3.37	49.86	0.86	48.62	1.14	49.4	12.44	49.98
PSD	3.62	60.7	5.7	63.43	3.85	87.02	7.18	89.24	25.46	82.53
TP	0.19	60.7	2.08	63.43	0.41	87.02	3.76	89.24	23.43	82.07
TF_Energy	85.58	59.88	87.59	62.03	85.8	85.42	89.18	86.51	108.61	79.53
Combined all Time-domain features (MAV, RMS, VAR, WL, ZC, SSC, WAMP)	6.77	67.49	9.17	76.3	6.53	91.07	12.08	91.7	19.46	90.49
Combined all Frequency-domain features (MNF, MDF, PF, PSD, TP)	14.71	63.66	18.1	69.82	14.57	79.06	21.47	86.82	26.83	81.29
Combined all Time, Frequency, Time-Frequency-domain features (MAV, RMS, VAR, WL, ZC, SSC, WAMP, MNF, MDF, PF, PSD, TP, WT)	107	70.06	110.16	80.43	106.66	86.59	116.14	90.21	115.13	88.42
Average	14.72	57.80	16.99	60.88	**14.85**	**75.50**	18.26	78.46	36.49	72.94

Table. 06 Comparison of classifiers based on accuracy and processing time.

In terms of feature-wise comparison, time-domain features (MAV, RMS, VAR, WL, SSC) are generally led to higher accuracy in all models, particularly KNN and Random Forest. In frequency-domain features, power spectral density (PSD) and total power (TP) are performed well in all models, specifically in Random Forest (89.24%) and KNN (87.02%). Finally, combined feature sets (all time-domain, all frequency-domain, and all

time, frequency, and time-frequency domain) provide better accuracy in most of the models, but processing times are higher. See figure 22 for feature accuracy and processing time for different classifiers.

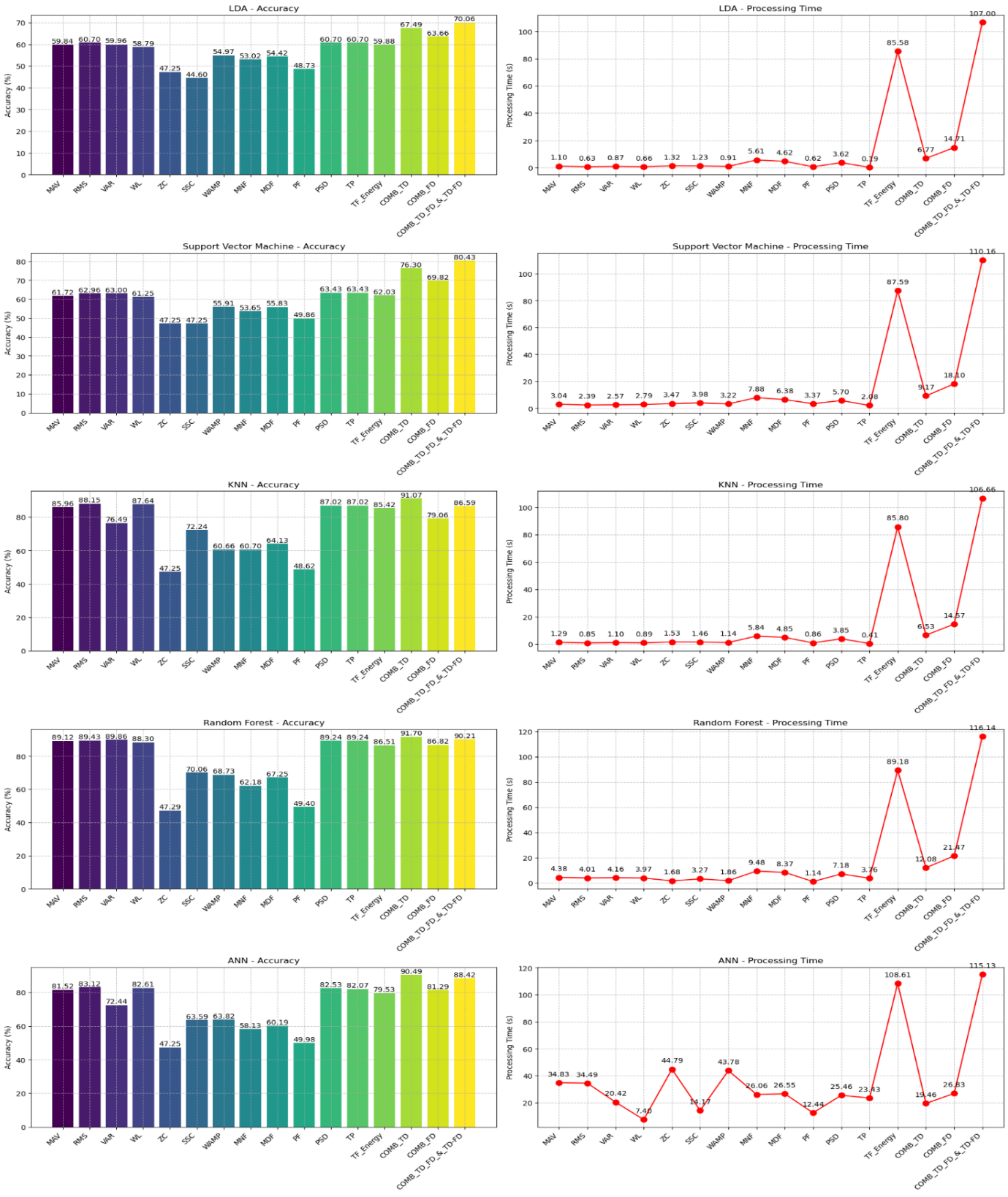


Figure. 22 Feature accuracy and processing time for different classifiers.

Overall, the best-performing model is K-Nearest Neighbour (KNN), which stands out for both accuracy and processing time, and it consistently achieves good accuracy in all models with considerable processing time. Random Forest performs strongly in all models but has a slightly higher processing time. Artificial Neural Networks (ANN) achieve higher accuracy, but the cost of processing time is very high and less efficient in real-time applications. Finaly, linear discriminant analysis (LDA) goes through faster in all models but struggles with lower accuracy.

b. Find the best model based on mean accuracy and standard deviation.

This session focuses on determining optimal model and feature selection based on mean accuracy and standard deviation, with higher mean accuracy indicating better prediction performance and lower standard deviation indicating consistency. In our scenario, KNN and Random Forest achieve the highest mean accuracy consistently for most of the features. Particularly, the combined time-domain features of KNN reach 91.70% and Random Forest achieve 90.74%. The LDA and ANN generally perform lower than KNN and Random Forest, and their mean accuracy ranges in between 60 and 70 percent. SVM provides moderate performance, and the highest mean accuracy reached 81.74% at the combined time, frequency, and time-frequency domain feature. The noticeable point is that some features, such as MAV, RMS, PSD, TP, and TF-Energy features, are consistently achieving more than 80 percent mean accuracy in KNN and Random Forest models. In the case of standard deviation, KNN and Random Forest exhibit lower standard deviations, indicating more consistent performance. Th range of standard deviation of KNN mostly under 1.5, with the lowest at 0.54. Overall KNN and Random-Forest are performed better than other classifiers.

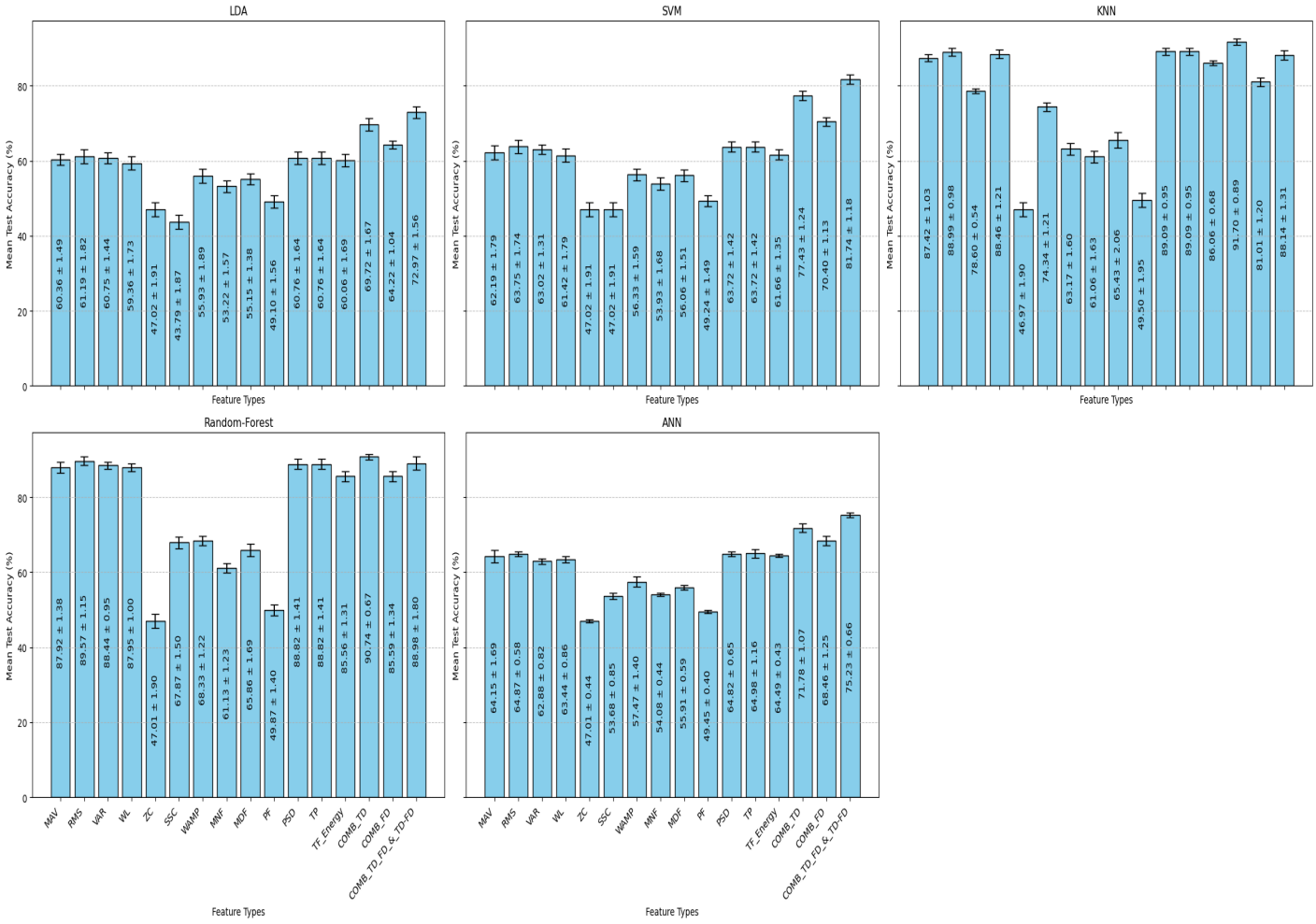


Figure 23 Mean accuracy and standard deviation for different classifiers.

c. Find the best model based on mean classification error.

The mean classification error in different models, the KNN and Random Forest, has the lowest error rate (23.19% and 22.35%), and it indicates the best-performing models See figure 24 for classification error comparison for different classifiers. Finally, see Table.07 for the best classifier models and their best features.

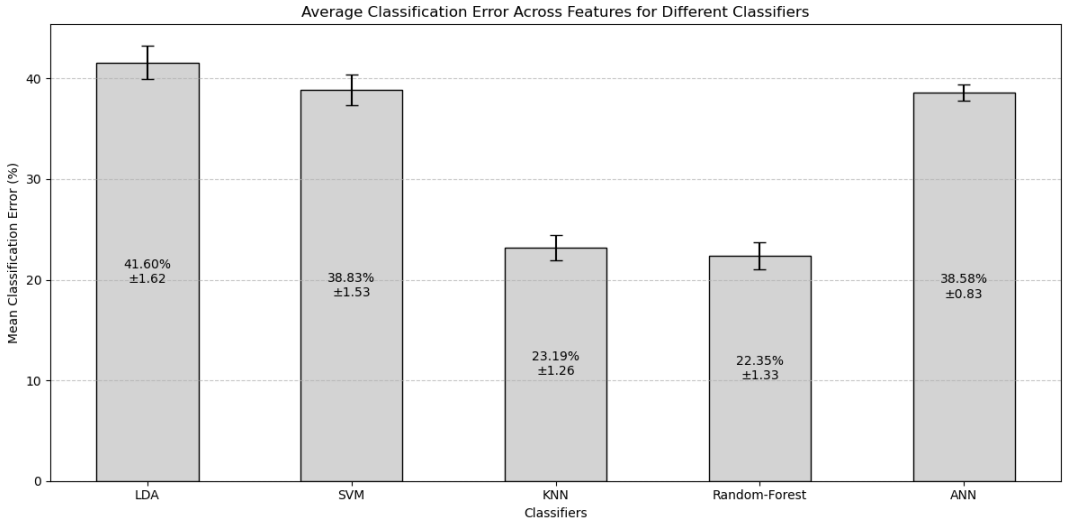


Figure 24 Classification error comparison for different classifiers

Top Models and Their Best Features

KNN	Mean Accuracy (%)	Std Deviation	Random-Forest	Mean Accuracy (%)	Std Deviation
Combined Time-Domain Feature	91.70	0.89	Combined Time-Domain Feature	90.74	0.67
Power Spectral Density Feature (PSD)	89.09	0.95	Root Mean Square Feature (RMS)	89.57	1.15
Total Power Feature (TP)	89.09	0.95	Combined Time, frequency, and Time-Frequency Domain	88.98	1.80
Wavelength Feature (WL)	88.82	1.41	Total Power Feature (TP)	88.82	1.41
Mean Absolute Value (MAV)	87.42	1.03	TF Energy Feature	88.82	1.31

Table 07 Best classification models and their beast features.

d. Confusion matrix for KNN model with their best features.

The confusion matrix helps to identify which classes perform well or not in their classifier model. Random Forest indicates good performance in class 0 (6019) and weakest true positive in class 10 (317). Classes 2 and 7 have also provided better true positives for effective classification. On the other hand, classes 10 and 11 need more attention to improvement. The KNN classifier model performed significantly well in class 0, showing a high number of true positives (5934). The weakest performance was class 10, with the fewest true positives (277). Finally, Random Forest is slightly better than KNN, especially in classes with lower true positives. See Figure 25 confusion matrix for best classifier models.

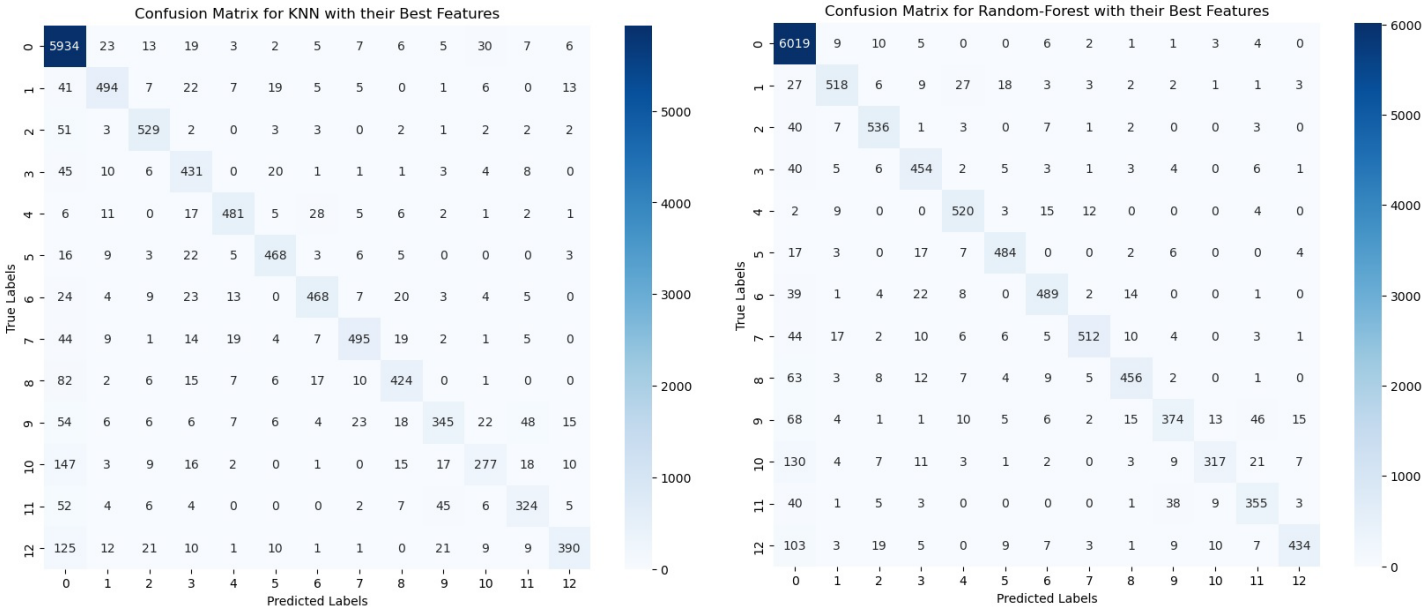


Figure. 25 Confusion matrix for best classifier models.

Part. C Individual Subject Analysis (S1_A1_E1 to S27_A1_E1).

1. Discover the best features for each subject.

In this part, deal with individual subjects (1 to 27) and find the accuracy in all time- domains and frequency-domains. The combined both time and frequency domain features into a single data set to find the accuracy in KNN and Random Forest models. See table 08 for the best features in individual subjects. Based on KNN and Random Forest mean testing accuracy comparison, the Random Forest classifier has a higher than k-NN (Random Forest: 93.18% and KNN: 90.20%). See figure 26 for mean accuracy for combined time and frequency domain features in individual subjects.

Subjects	Features name	Random forest Testing Accuracy	KNN Testing Accuracy
S1	Combined_Time_and_Frequency_Domain_Feature	95.91	94.55
S2	Combined_Time_and_Frequency_Domain_Feature	95.69	93.97
S3	Combined_Time_and_Frequency_Domain_Feature	95.95	93.69
S4	Combined_Time_and_Frequency_Domain_Feature	92.47	89.96
S5	Combined_Time_and_Frequency_Domain_Feature	92.86	84.92
S6	Combined_Time_and_Frequency_Domain_Feature	93.64	93.22
S7	Combined_Time_and_Frequency_Domain_Feature	95.30	91.45
S8	Combined_Time_and_Frequency_Domain_Feature	96.83	95.93
S9	Combined_Time_and_Frequency_Domain_Feature	89.18	83.98
S10	Combined_Time_and_Frequency_Domain_Feature	94.89	89.79
S11	Combined_Time_and_Frequency_Domain_Feature	88.48	84.77
S12	Combined_Time_and_Frequency_Domain_Feature	92.83	87.34
S13	Combined_Time_and_Frequency_Domain_Feature	95.77	93.90
S14	Combined_Time_and_Frequency_Domain_Feature	92.31	88.89
S15	Combined_Time_and_Frequency_Domain_Feature	96.14	92.27
S16	Combined_Time_and_Frequency_Domain_Feature	93.27	89.69
S17	Combined_Time_and_Frequency_Domain_Feature	94.55	92.73
S18	Combined_Time_and_Frequency_Domain_Feature	89.47	80.70
S19	Combined_Time_and_Frequency_Domain_Feature	93.99	92.70
S20	Combined_Time_and_Frequency_Domain_Feature	93.30	93.30
S21	Combined_Time_and_Frequency_Domain_Feature	97.10	93.78
S22	Combined_Time_and_Frequency_Domain_Feature	89.52	88.31
S23	Combined_Time_and_Frequency_Domain_Feature	90.64	87.66
S24	Combined_Time_and_Frequency_Domain_Feature	96.09	94.78
S25	Combined_Time_and_Frequency_Domain_Feature	92.08	90.42
S26	Combined_Time_and_Frequency_Domain_Feature	92.14	89.08
S27	Combined_Time_and_Frequency_Domain_Feature	85.37	83.74

Table 08 Best features in individual subjects.

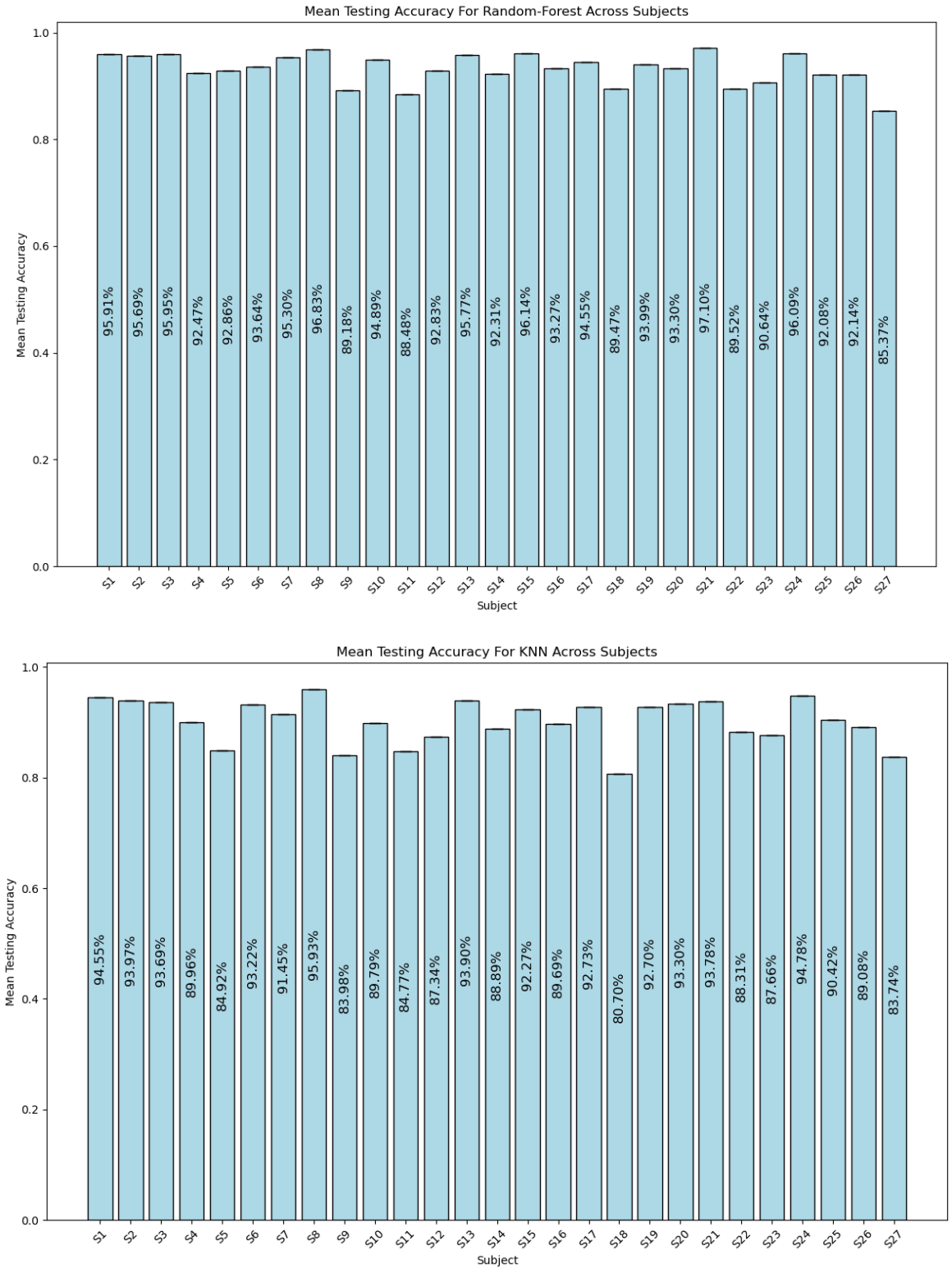


Figure 26 Mean accuracy for combined time and frequency domain feature in individual subjects.

2. Find the best classification model for individual subject.

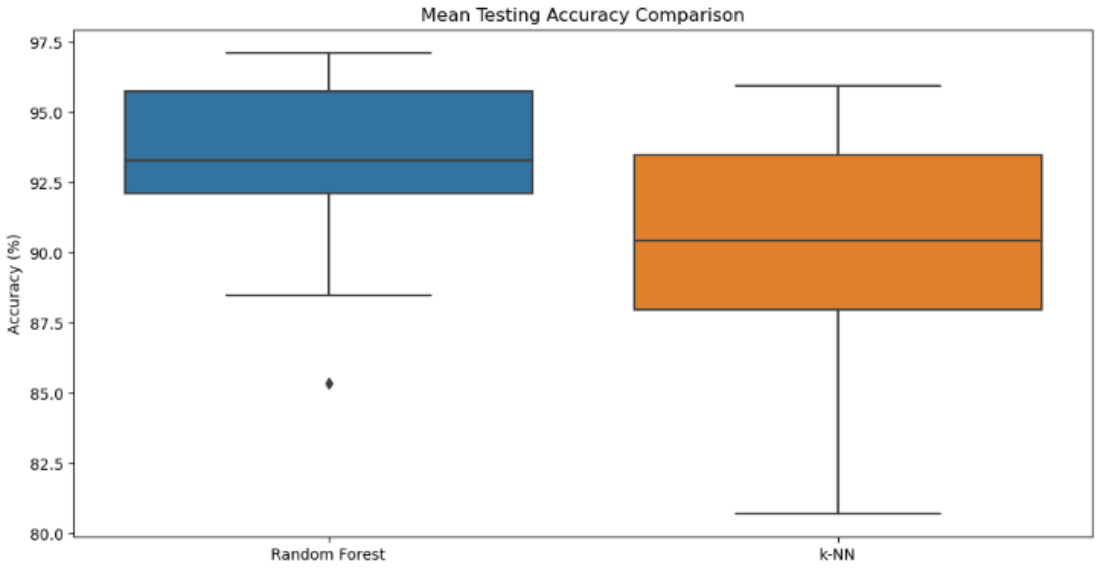


Figure 27 Mean testing accuracy for Random-forest and KNN

During all subjects, Random Forest performed slightly better than the KNN model, and the difference in accuracy ranged from 0.90 percent to 5.10 percent. The top performing subject is 21, and it achieves 97.10 percent in the random forest classifier and 95.93 percent in subject 8. In subjects 8, 15, and 24, they also indicate better accuracy just above the 96 percent in the Random Forest classifier, and similarly, subjects 24 and 1 achieve 94.78 and 94.55 percent, respectively. The top subjects based on accuracy in both classifiers are 1, 2, 3, 8, 13, 21, and 24. The lowest accuracy (85.37%) achieved in subject 27 at Random Forest and KNN is subject 18 (80.70%). See table 08 for best features in individual subjects. The high-performance subjects (S21, S8, S24, S15, and S3) are exceedingly above 95 percent. The highest accuracy subject remains the same as 27 (97.10%). Several subjects, including S13, S1, S2, and S17, are performed in moderate as compared to the highest accuracy subject (S27).

3. Perform a statistical analysis test on three different features.

This statistical non-parametric Friedman test [56] was used to detect differences in treatments across multiple test attempts. For this experiment, I chose three different features (wavelength (WL), total power (TP), and time-frequency energy (TF energy)). See table 09 for testing accuracy and ranking for three different features. Then, perform the Friedman test to statistically compare the accuracy results across the three features. The results indicated a Friedman test statistic of 31.98 and a **p-value of 1.14×10^{-7},** demonstrating a significant difference in performance between the features. A lower mean rank indicates better performance; in the Friedman test and ranking analysis, the tf-energy feature (time-frequency domain) has the best average rank (1.22), meaning it performed the best as compared to the other features across all subjects. Overall, the lowest p-value indicates that the observed differences in performance are statistically significant and not due to random chance. This significant result provides tf-energy features, which is the best performance feature.

	Testing accuracy			Ranking		
Subjects	**wl_features**	**tp_features**	**tf_energy_features**	**wl_features**	**tp_features**	**tf_energy_features**
S1	95.45	93.18	92.73	3	2	1
S2	92.24	95.26	91.38	2	3	1
S3	95.95	95.95	93.69	2.5	2.5	1
S4	90.80	93.31	90.38	2	3	1
S5	88.10	93.25	84.52	2	3	1
S6	89.41	94.92	87.71	2	3	1
S7	93.16	94.44	90.17	2	3	1
S8	95.93	96.38	94.57	2	3	1
S9	87.88	88.31	83.55	2	3	1
S10	94.89	94.47	92.77	3	2	1
S11	87.24	89.71	83.95	2	3	1
S12	85.23	90.72	83.54	2	3	1
S13	92.49	96.71	92.49	1.5	3	1.5
S14	85.47	90.60	81.62	2	3	1
S15	93.56	95.71	90.99	2	3	1
S16	91.48	91.93	91.48	1.5	3	1.5
S17	95.00	95.45	95.00	1.5	3	1.5
S18	86.84	88.60	79.82	2	3	1
S19	90.13	93.56	87.98	2	3	1
S20	93.75	91.52	90.18	3	2	1
S21	95.44	95.02	95.44	2.5	1	2.5
S22	87.90	89.92	85.08	2	3	1
S23	86.81	91.06	86.38	2	3	1
S24	93.04	95.22	90.43	2	3	1
S25	91.25	93.75	90.00	2	3	1
S26	89.96	88.65	90.83	2	1	3
S27	81.71	87.40	83.33	1	3	2
Mean Ranking				**2.06**	**2.72**	**1.22**

Table. 09 Testing accuracy and ranking for three different features.

Part. D Propose the hybrid model.

This hybrid approach is experimented with in combining 27 subjects into a single dataset and extracting the best features such as wavelength, total power, and time-frequency energy. Then, after these features are extracted, and combine into a single feature and named as hybrid features. It contains multiple information from various domains. Finally, calculate the testing and training accuracy using KNN and Random Forest. See table 10 for an accuracy comparison for hybrid models.

Metric	KNN	Random-Forest
Testing Accuracy	88.89%	90.49%
Training Accuracy	93.33%	100.00%
Macro Average Precision	0.85	0.9
Macro Average Recall	0.82	0.84
Macro Average F1-Score	0.83	0.86
Weighted Average Precision	0.89	0.91
Weighted Average Recall	0.89	0.9
Weighted Average F1-Score	0.89	0.9

Table. 10 Accuracy comparison for hybrid models

Random Forest outperformed KNN in accuracy, precision, recall, and F1-score metrics, but has perfect training accuracy (100%), potentially indicating overfitting. KNN computation time is more efficient in real-time scenarios. A combined hybrid approach with 27 subjects helps learn more robust patterns but increases training data size. This approach may capture variations and simplify the model development process.

Conclusions

In conclusion, the key conclusions are provided after multiple experiments for each target. For window dimensions ranging from 500 to 250, it provides a better balance of accuracy and calculation time. Second, in terms of filtering, model 4 is the most important fulfilled condition. It gave better signal stability than the other model conditions (1, 2, and 3).

In Part B, undertake a thorough examination of feature comparison across 27 datasets. To conduct statistical analysis, use mean, median, and standard deviation to categorise characteristics as best (SSC, WL, TP, TF-Energy), intermediate (WAMP, RMS, MNF, MDF, PSD), or least informative (MAV, VAR, ZC, PF). Second, count the number of outliers among the excellent characteristics that SSC provides. Finally, conduct experiments to choose the best classifier model based on accuracy and processing time in various features. K-Nearest Neighbours (KNN) and Random Forest (RF) performed well.

Part C: experiment with the most influential features and models across 27 different individuals. Random Forest is performed well in the combined time and frequency domain feature across all subjects (accuracy ranges from 85.37% to 97.10%). Second, Friedman statistical test analysis on the best features (WL, TP, and TF-Energy). The TF-energy feature obtained the highest average rank (1.22), outperforming the other features in all subjects. Finally, in Part D, we proposed a hybrid model that incorporates the best features and classifiers. Random Forest exceeded KNN in terms of accuracy (90.49%) as well as precision, recall, and F1-score measures (both macro and weighted averages); however, it took little more time to compute.

Acknowledgement

I express my heartfelt gratitude to the following people and institutions for their invaluable contributions to this work:

- Dr. Simon Stuttaford provided excellent assistance and insightful criticism throughout the project's exploration.
- Dr. Rishad Shafik and Dr. Kabita Adhikari for their valuable insights into machine learning fundamentals and inspiration for choosing machine learning based project.
- Dr. Mathew Dyson for his significant suggestions and advice in improving the entire project.
- Dr. Andrew Smith for assisting with the technical documentation template format and providing helpful comments.
- With gratitude to the management team at Newcastle University for providing a top-notch computer lab in the Stephenson Building and granting permission to operate after hours.
- Finally, my friends and family support and encourage me to accomplish my dissertation work.

Reference

[1] M. Atzori, A. Gijsberts, C. Castellini, B. Caputo, A. G. Hager, S. Elsig, G. Giatsidis, F. Bassetto, and H.Müller, "Electromyography data for non-invasive naturally-controlled robotic hand prostheses," *Sci. Data*, vol. 1, Art. no. 140053, Dec. 2014, doi: 10.1038/sdata.2014.53.

[2] D. J. Atkins, "Epidemiologic overview of individuals with upper-limb loss and their reported research priorities," *J. Prosthetics Orthotics*, vol. 8, no. 1, pp. 2–11, 1996.

[3] A. Gijsberts, M. Atzori, C. Castellini, B. Caputo, A. Mouhoubi, S. P. D. Paredes, and H. Müller, "Characterization of a benchmark database for myoelectric movement classification," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 22, no. 5, pp. 1065–1074, Sep. 2014. [Online]. Available: https://ninapro.hevs.ch/instructions/DB1.html

[4] K. M. K. Okkesim and E. Sezgin, "Hand movements classification using feature extraction from surface electromyography," ResearchGate, 2023. [Online]. Available: https://www.researchgate.net/publication/372160313_Hand_Movements_classification_using_feature_extraction_from_Surface_Electromyography

[5] C. Ouyang, L. Cai, B. Liu, et al., "An improved wavelet threshold denoising approach for surface electromyography signal," *EURASIP J. Adv. Signal Process.*, vol. 2023, no. 108, 2023, doi: 10.1186/s13634-023-01066-3.

[6] D. B. Kulkarni and G. R. Udupi, "ANN-based SVC switching at distribution level for minimal-injected harmonics," *IEEE Trans. Power Del.*, vol. 25, no. 3, pp. 1978-1985, Jul. 2010, doi: 10.1109/TPWRD.2010.2040293.

[7] B. Kundu and D. Subarram Naidu, "Classification and feature extraction of different hand movements from EMG signal using machine learning-based algorithms," in *Proc. Int. Conf. Electr. Commun. Comput. Eng. (ICECCE)*, Kuala Lumpur, Malaysia, 2021, pp. 1-5, doi: 10.1109/ICECCE52056.2021.9514134.

[8] P. K. Singh, R. Sarkar, and M. Nasipuri, "Significance of non-parametric statistical tests for comparison of classifiers over multiple datasets," *Int. J. Comput. Sci. Math.*, vol. 7, no. 5, pp. 410–442, 2016.

[9] A. Bhattacharya, A. Sarkar, and P. Basak, "Time domain multi-feature extraction and classification of human hand movements using surface EMG," in *Proc. Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Coimbatore, India, 2017, pp. 1-5, doi: 10.1109/ICACCS.2017.8014594.

[10] O. W. Samuel, M. G. Asogbon, Y. Geng, A. H. Al-Timemy, S. Pirbhulal, N. Ji, S. Chen, P. Fang, and G. Li, "Intelligent EMG pattern recognition control method for upper-limb multifunctional prostheses: Advances, current challenges, and future prospects," *IEEE Access*, vol. 7, pp. 10150–10165, 2019.

[11] C. Spiewak, "A comprehensive study on EMG feature extraction and classifiers," *Open Access J. Biomed. Eng. Biosci.*, vol. 1, pp. 17–26, 2018.

[12] R. A. R. C. Gopura, D. S. V. Bandara, J. M. P. Gunsekara, and T. S. S. Jayawardane, "Recent trends in EMG-based control methods for assistive robots," in *Electrodiagnosis in New Frontiers of Clinical Research*, H. Turker, Ed., London, U.K.: IntechOpen, 2013, pp. 237–268.

[13] Otto Bock HealthCare, "MYOBOCK® electrodes for 6/7.2-volt systems," Technical Data Sheet, pp. 6.82-6.83, 2024. [Online]. Available: https://forum.pololu.com/uploads/default/original/2X/1/17374ecfa9119bb3c440a03f7e722aeedd414002.pdf. [Accessed: Aug. 15, 2024].

[14] B. Hudgins, P. Parker, and R. N. Scott, "A new strategy for multifunction myoelectric control," *IEEE Trans. Biomed. Eng.*, vol. 40, no. 1, pp. 82-94, Jan. 1993.

[15] A. Phinyomark, C. Limsakul, and P. Phukpattaranont, "A novel feature extraction for robust EMG pattern recognition," *J. Comput.*, vol. 1, no. 1, pp. 71-80, Dec. 2009.

[16] M. A. Oskoei and H. Hu, "Myoelectric control systems—A survey," *Biomed. Signal Process. Control*, vol. 2, no. 4, pp. 275-294, Oct. 2007.

[17] R. G. Willison, "Analysis of electrical activity in healthy and dystrophic muscle in man," *J. Neurol. Neurosurg. Psychiatry*, vol. 26, no. 4, pp. 344-352, Aug. 1963.

[18] A. Phinyomark, P. Phukpattaranont, and C. Limsakul, "Feature reduction and selection for EMG signal classification," *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7420-7431, Jun. 2012.

[19] P. D. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Trans. Audio Electroacoust.*, vol. 15, no. 2, pp. 70-73, Jun. 1967.

[20] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowl. Discov.*, vol. 2, pp. 121-167, Jun. 1998.

[21] P. S. Addison, "Wavelet transforms and the ECG: A review," *Physiol. Meas.*, vol. 26, no. 5, pp. R155-R199, Oct. 2005.

[22] A. Phinyomark, P. Phukpattaranont, and C. Limsakul, "Feature reduction and selection for EMG signal classification," *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7420-7431, Jun. 2012.

[23] J. S. Bendat and A. G. Piersol, *Random Data: Analysis and Measurement Procedures*, 4th ed., Hoboken, NJ, USA: Wiley, 2011.

[24] N. Parajuli, N. Sreenivasan, P. Bifulco, M. Cesarelli, S. Savino, V. Niola, D. Esposito, T. J. Hamilton, G. R. Naik, U. Gunawardana, and G. D. Gargiulo, "Real-time EMG-based pattern recognition control for hand prostheses: A review on existing methods, challenges and future implementation," *Sensors*, vol. 19, no. 20, pp. 1-27, Oct. 2019, doi: 10.3390/s19204596.

[25] R. H. Chowdhury, M. B. Reaz, M. A. Ali, A. A. Bakar, K. Chellappan, and T. G. Chang, "Surface electromyography signal processing and classification techniques," *Sensors (Basel)*, vol. 13, no. 9, pp. 12431-12466, Sep. 2013, doi: 10.3390/s130912431.

[26] H. P. B., "Electrode noise and its effects on the EMG signal," *J. Biomed. Eng.*, vol. 31, pp. 232-237, 2022.

[27] A. Kumar et al., "High-quality electrode design and noise reduction in EMG signals," *Biomed. Signal Process. Control*, vol. 42, pp. 56-65, 2018.

[28] X. Navarro, T. B. Krueger, N. Lago, S. Micera, T. Stieglitz, and P. Dario, "A critical review of interfaces with the peripheral nervous system for the control of neuroprostheses and hybrid bionic systems," *J. Peripher. Nerv. Syst.*, vol. 10, pp. 229–258, 2005.

[29] M. B. I. Reaz, M. S. Hussain, and F. Mohd-Yasin, "Techniques of EMG signal analysis: Detection, processing, classification, and applications," *Biol. Proced. Online*, vol. 8, pp. 11-35, 2006.

[30] J. Liu et al., "Adaptive filtering techniques for reducing movement artifacts in EMG signals," *IEEE Trans. Biomed. Eng.*, vol. 60, no. 6, pp. 1537-1546, Jun. 2013.

[31] C. B. Phinyomark et al., "Wavelet transform for motion artifact reduction in EMG signals," *J. Electromyogr. Kinesiol.*, vol. 21, no. 6, pp. 1018-1027, Dec. 2011.

[32] M. J. G., "Adaptive filters for electromagnetic noise reduction in EMG signals," *J. Signal Process.*, vol. 30, no. 4, pp. 273-281, 2019.

[33] T. F. Yang et al., "Laguerre filter for power-line interference removal in EMG signals," *IEEE Trans. Biomed. Eng.*, vol. 56, no. 7, pp. 1685-1692, Jul. 2009.

[34] A. Lowery et al., "Minimizing crosstalk in surface electromyography: Electrode size and placement," *IEEE Trans. Biomed. Eng.*, vol. 53, no. 6, pp. 1216-1223, Jun. 2006.

[35] S. L. Mezzarane et al., "Mathematical models for crosstalk reduction in EMG signal analysis," *J. Electromyogr. Kinesiol.*, vol. 18, no. 5, pp. 832-842, Oct. 2011.

[36] S. R. Hemingway et al., "Effects of tissue thickness on surface EMG signal amplitude," *IEEE Trans. Biomed. Eng.*, vol. 57, no. 9, pp. 2256-2263, Sep. 2010.

[37] Y. Z. Zhang et al., "High-pass filters for reducing internal noise in EMG signals," *Biomed. Signal Process. Control*, vol. 16, pp. 134-142, Jun. 2015.

[38] C. Hu et al., "High-pass filtering for removing ECG artifacts in surface EMG," *J. Biomed. Eng.*, vol. 32, no. 7, pp. 1347-1355, Jul. 2014.

[39] A. D. Webster et al., "Common-mode rejection for minimizing ECG contamination in EMG signals," *IEEE Trans. Biomed. Eng.*, vol. 62, no. 8, pp. 2012-2019, Aug. 2015.

[40] L. Chen, X. Liu, and S. Zhang, "Real-time muscle activity monitoring using sliding window techniques," *J. Electromyogr. Kinesiol.*, vol. 55, pp. 102-110, Oct. 2020.

[41] J. Zhang, S. Wei, and Y. Sun, "Dynamic Windowing Approaches for Enhanced EMG Signal Analysis," *Biomed. Signal Process. Control*, vol. 75, p. 103556, 2022.

[42] J. Kim, S. Park, and H. Lee, "Optimizing Sliding Window Parameters for Efficient EMG Signal Classification," *IEEE Trans. Biomed. Eng.*, vol. 68, no. 5, pp. 1342-1352, May 2021.

[43] Y. Li, T. Zhang, and J. Wang, "Effectiveness of Overlapping Sliding Windows in EMG Signal Processing," *J. Biomed. Sci. Eng.*, vol. 12, no. 7, pp. 1-12, Jul. 2019.

[44] X. Yang, Z. Xu, and L. Huang, "Adaptive Sliding Window Techniques for EMG Signal Analysis," *IEEE Access*, vol. 9, pp. 123456-123467, 2021.

[45] M. Atzori, A. Gijsberts, C. Castellini, B. Caputo, A. G. Hager, S. Elsig, ... and H. Muller, "Electromyography data for non-invasive naturally-controlled robotic hand prostheses," *Sci. Data*, vol. 1, no. 1, pp. 1-13, 2014.

[46] Y. Du, W. Jin, W. Wei, Y. Hu, and W. Geng, "Surface EMG-based inter-session gesture recognition enhanced by deep domain adaptation," *Sensors*, vol. 17, no. 3, p. 458, Mar. 2017.

[47] K. Englehart, B. Hudgins, and P. A. Parker, "A wavelet-based continuous classification scheme for multifunction myoelectric control," *IEEE Trans. Biomed. Eng.*, vol. 48, no. 3, pp. 302-311, Mar. 2001.

[48] A. Phinyomark, R. N. Khushaba, and E. Scheme, "Feature extraction and selection for myoelectric control based on wearable EMG sensors," *Sensors*, vol. 18, no. 5, p. 1615, May 2018.

[49] H. Chen, R. Tong, M. Chen, Y. Fang, and H. Liu, "A hybrid CNN-SVM classifier for hand gesture recognition with surface EMG signals," in *Proc. 2018 Int. Conf. Machine Learning Cybernetics (ICMLC)*, Chengdu, China, 2018, pp. 619-624, doi: 10.1109/ICMLC.2018.8526976.

[50] M. A. Oskoei and H. Hu, "Support vector machine-based classification scheme for myoelectric control applied to upper limb," *IEEE Trans. Biomed. Eng.*, vol. 55, no. 8, pp. 1956-1965, Aug. 2008.

[51] R. Rajapriya, K. Rajeswari, and S. J. Thiruvengadam, "Deep learning and machine learning techniques to improve hand movement classification in myoelectric control system," Dept. Electron. Commun., Thiagarajar College of Engineering, Madurai, India.

[52] M. A. Oskoei and H. Hu, "Support Vector Machine-Based Classification Scheme for Myoelectric Control Applied to Upper Limb," *IEEE Trans. Biomed. Eng.*, vol. 55, no. 8, pp. 1956-1965, Aug. 2008, doi: 10.1109/TBME.2008.919734.

[53] Y. Narayan, "sEMG signal classification using k-NN classifier with FD and TFD features," *J. Electromyogr. Kinesiol.*, vol. 55, no. 3, pp. 101-110, Oct. 2020, doi: 10.1016/j.jelekin.2020.102345.

[54] M. Fındık, Ş. Yılmaz, and M. Koseoglu, "Random Forest Classification of Finger Movements using Electromyogram (EMG) Signals," in *2020 IEEE SENSORS*, Rotterdam, Netherlands, 2020, pp. 1-4, doi: 10.1109/SENSORS47125.2020.9278619.

[55] U. Baspinar, H. S. Varol, and K. Yildiz, "Classification of hand movements by using artificial neural network," in *2012 Int. Symp. Innovations Intell. Syst. Appl.*, Trabzon, Turkey, 2012, pp. 1-4, doi: 10.1109/INISTA.2012.6247014.

[56] S. W. Scheff, *Fundamental Statistical Principles for the Neurobiologist: A Survival Guide*. Cambridge, MA, USA: Academic Press, 2016, ch. 8, pp. 157–182. doi: 10.1016/B978-0-12-804753-8.00008-7.

Appendix

1: MATLAB code data combination.

```
% Start overall timing
startTime = datetime('now');
```

```
% Clear the Command Window
clc
% Load the .mat file in each subjects or participiants(1 to 27)
data_01 = load('S1_A1_E1.mat');
data_02 = load('S2_A1_E1.mat');
data_03 = load('S3_A1_E1.mat');
data_04 = load('S4_A1_E1.mat');
data_05 = load('S5_A1_E1.mat');
data_06 = load('S6_A1_E1.mat');
data_07 = load('S7_A1_E1.mat');
data_08 = load('S8_A1_E1.mat');
data_09 = load('S9_A1_E1.mat');
data_10 = load('S10_A1_E1.mat');


data_11 = load('S11_A1_E1.mat');
data_12 = load('S12_A1_E1.mat');
data_13 = load('S13_A1_E1.mat');
data_14 = load('S14_A1_E1.mat');
data_15 = load('S15_A1_E1.mat');
data_16 = load('S16_A1_E1.mat');
data_17 = load('S17_A1_E1.mat');
data_18 = load('S18_A1_E1.mat');
data_19 = load('S19_A1_E1.mat');
data_20 = load('S20_A1_E1.mat');


data_21 = load('S21_A1_E1.mat');
data_22 = load('S22_A1_E1.mat');
data_23 = load('S23_A1_E1.mat');
data_24 = load('S24_A1_E1.mat');
data_25 = load('S25_A1_E1.mat');
data_26 = load('S26_A1_E1.mat');
data_27 = load('S27_A1_E1.mat');
```

```
% Display each subjects or participiants size (1 to 27)


for i = 1:27
    % Generate the filename for the current subject
    filename = sprintf('S%d_A1_E1.mat', i);
```

```
    % Load the .mat file for the current subject
    data = load(filename);

    % Print the size of each data matrix
    fprintf('Subject %02d:\n', i);
    fprintf('  EMG data size: %s\n', mat2str(size(data.emg)));
    fprintf('  Stimulus data size: %s\n', mat2str(size(data.stimulus)));
    fprintf('  Restimulus data size: %s\n', mat2str(size(data.restimulus)));
    fprintf('  Repetition data size: %s\n', mat2str(size(data.repetition)));
    fprintf('  Rerepetition data size: %s\n', mat2str(size(data.rerepetition)));
    fprintf('\n');
end
```

```
% Extract variables in each subjects or participiants(1 to 27)
% Loop through each subject from 1 to 27
for i = 1:27
    % Generate the base variable name for the current subject
    % For example, for i=1, data_var_name will be 'data_01'
    data_var_name = sprintf('data_%02d', i);


    % Generate the names for the extracted variables
    % For example, for i=1, emg_var_name will be 'emg_data_01'
    emg_var_name = sprintf('emg_data_%02d', i);
    stimulus_var_name = sprintf('stimulus_data_%02d', i);
    restimulus_var_name = sprintf('restimulus_data_%02d', i);
    repetition_var_name = sprintf('repetition_data_%02d', i);
    rerepetition_var_name = sprintf('rerepetition_data_%02d', i);


    % Extract and assign the EMG data using eval
    % For example, eval('emg_data_01 = data_01.emg;')
    eval([emg_var_name ' = ' data_var_name '.emg;']);
    % Extract and assign the stimulus data using eval
    % For example, eval('stimulus_data_01 = data_01.stimulus;')
    eval([stimulus_var_name ' = ' data_var_name '.stimulus;']);
    % Extract and assign the restimulus data using eval
    eval([restimulus_var_name ' = ' data_var_name '.restimulus;']);
    % Extract and assign the repetition data using eval
    eval([repetition_var_name ' = ' data_var_name '.repetition;']);
    % Extract and assign the rerepetition data using eval
    eval([rerepetition_var_name ' = ' data_var_name '.rerepetition;']);
end
```

```
% comine all data participiants(0 to 27)
% Preallocate cell arrays to store each type of data for all participants
emg_data_all = [];
stimulus_data_all = [];
restimulus_data_all = [];
repetition_data_all = [];
```

```
rerepetition_data_all = [];
% Loop through each participant
for i = 1:27
    % Generate the filename for the current participant
    filename = sprintf('S%d_A1_E1.mat', i);

    % Load the .mat file for the current participant
    data = load(filename);

    % Concatenate the data from the current participant to the combined matrices
    emg_data_all = [emg_data_all; data.emg];
    stimulus_data_all = [stimulus_data_all; data.stimulus];
    restimulus_data_all = [restimulus_data_all; data.restimulus];
    repetition_data_all = [repetition_data_all; data.repetition];
    rerepetition_data_all = [rerepetition_data_all; data.rerepetition];
end
% Combine all data into a single matrix
combined_data_all= [emg_data_all, stimulus_data_all, restimulus_data_all,
repetition_data_all, rerepetition_data_all];
% Print the size of the combined data matrix
fprintf('Combined data size of 1 to 27 Participiants: %s\n',
mat2str(size(combined_data_all)));
fprintf('  EMG data_all size: %s\n', mat2str(size(emg_data_all)));
fprintf('  Stimulus_data_all size: %s\n', mat2str(size(stimulus_data_all)));
fprintf('  Restimulus_data_all size: %s\n', mat2str(size(restimulus_data_all)));
fprintf('  Repetition_data_all size: %s\n', mat2str(size(repetition_data_all)));
fprintf('  Rerepetition_data_all size: %s\n', mat2str(size(rerepetition_data_all)));
```

```
% Define window size and increment
winsize = 500;  % Window size in samples (800, 500, 200, 150, 100, 50)
wininc = 250;   % Window increment in samples (400, 250, 100, 75, 50, 25)
```

```
% Initialize storage for each feature across all classes
all_mav_features = [];
all_rms_features = [];
all_var_features = [];
all_wl_features = [];
all_zc_features = [];
all_ssc_features = [];
all_wamp_features = [];


all_mnf_features = [];
all_mdf_features = [];
```

```
all_pf_features = [];
all_psd_features = [];
all_tp_features = [];


all_tf_energy_features = [];
```

```
% Initialize timing variables for each class
time_mav_class = 0;
time_rms_class = 0;
time_var_class = 0;
time_wl_class = 0;
time_zc_class = 0;
time_ssc_class = 0;
time_wamp_class = 0;
time_mnf_class = 0;
time_mdf_class = 0;
time_pf_class = 0;
time_psd_class = 0;
time_tp_class = 0;


time_tf_energy_class = 0;


% Initialize timing variables for all classes
total_time_mav = 0;
total_time_rms = 0;
total_time_var = 0;
total_time_wl = 0;
total_time_zc = 0;
total_time_ssc = 0;
total_time_wamp = 0;
total_time_mnf = 0;
total_time_mdf = 0;
total_time_pf = 0;
total_time_psd = 0;
total_time_tp = 0;
total_time_tf_energy = 0;
```

```
% Define the save path
save_path = 'C:\Users\c3054367\OneDrive - Newcastle
University\lab\July\25072024\Combined\Matlab\Output_2507_500_250';
% Change this to your desired path


% Loop through each class from 0 to 12
```

```
for class_label = 0:12
    % Step 1: Filter the data where stimulus and restimulus match the class_label and
repetition equals rerepetition
    filtered_indices = find(stimulus_data_all == class_label & restimulus_data_all ==
class_label & repetition_data_all == rerepetition_data_all);

    % Apply filtering to the EMG data
    emg_filtered = emg_data_all(filtered_indices, :);

    % Step 2: Calculate the number of windows
    datasize = size(emg_filtered, 1);
    numwin = floor((datasize - winsize) / wininc) + 1;

    % Display the number of windows for the current class
    fprintf('Number of windows for class %d: %d\n', class_label, numwin);

    % Step 3: Allocate storage for features of the current class
    mav_features = zeros(numwin, size(emg_filtered, 2) + 1);  % MAV
    rms_features = zeros(numwin, size(emg_filtered, 2) + 1);  % RMS
    var_features = zeros(numwin, size(emg_filtered, 2) + 1);  % VAR
    wl_features = zeros(numwin, size(emg_filtered, 2) + 1);  % WL
    zc_features = zeros(numwin, size(emg_filtered, 2) + 1);  % ZC
    ssc_features = zeros(numwin, size(emg_filtered, 2) + 1);  % SSC
    wamp_features = zeros(numwin, size(emg_filtered, 2) + 1);  % WAMP


    mnf_features = zeros(numwin, size(emg_filtered, 2) + 1);  % MNF
    mdf_features = zeros(numwin, size(emg_filtered, 2) + 1);  % MDF
    pf_features = zeros(numwin, size(emg_filtered, 2) + 1);  % PF
    psd_features = zeros(numwin, size(emg_filtered, 2) + 1);  % PSD
    tp_features = zeros(numwin, size(emg_filtered, 2) + 1);  % TP
    vcf_features = zeros(numwin, size(emg_filtered, 2) + 1);  % VCF


    tf_energy_features = zeros(numwin, size(emg_filtered, 2) + 1);  % Time-frequency
domain features
```

```
    % Compute features for each window
    for i = 1:numwin
        % Extract current window
        curwin = emg_filtered((i-1)*wininc + 1 : (i-1)*wininc + winsize, :);


        % Start measuring time for MAV feature extraction
        tic;
        % Time-domain features
        % 1. Mean Absolute Value (MAV)
        mav_features(i, 1:end-1) = mean(abs(curwin));
        time_mav = toc;
        time_mav_class = time_mav_class + time_mav;


        % 2. Root Mean Square (RMS)
```

```
tic;
rms_features(i, 1:end-1) = sqrt(mean(curwin.^2));
time_rms = toc;
time_rms_class = time_rms_class + time_rms;

% 3. Variance (VAR)
tic;
var_features(i, 1:end-1) = var(curwin);
time_var = toc;
time_var_class = time_var_class + time_var;


% 4. Waveform Length (WL)
tic
wl_features(i, 1:end-1) = sum(abs(diff(curwin)));
time_wl = toc;
time_wl_class = time_wl_class + time_wl;


% 5. Zero Crossing (ZC)
tic;
zc_features(i, 1:end-1) = sum(diff(sign(curwin)) ~= 0);
time_zc = toc;
time_zc_class = time_zc_class + time_zc;

% 6. Slope Sign Change (SSC)
tic;
ssc_features(i, 1:end-1) = sum(diff(sign(diff(curwin))) ~= 0);
time_ssc = toc;
time_ssc_class = time_ssc_class + time_ssc;


% 7. Willison Amplitude (WAMP)
tic;
threshold = 0.1;  % Example threshold value
wamp_features(i, 1:end-1) = sum(abs(diff(curwin)) > threshold);
time_wamp = toc;
time_wamp_class = time_wamp_class + time_wamp;
```

```
% Frequency-domain features
% Compute the Fast Fourier Transform (FFT)
fft_result = fft(curwin);
P2 = abs(fft_result / winsize);
P1 = P2(1:winsize/2+1, :);
P1(2:end-1, :) = 2*P1(2:end-1, :);
f = (0:(winsize/2)) / winsize;

% 1. Mean Frequency (MNF)
tic;
mnf_features(i, 1:end-1) = meanfreq(P1, f);
time_mnf = toc;
time_mnf_class = time_mnf_class + time_mnf;
```

```
        % 2. Median Frequency (MDF)
        tic;
        mdf_features(i, 1:end-1) = medfreq(P1, f);
        time_mdf = toc;
        time_mdf_class = time_mdf_class + time_mdf;

        % 3. Peak Frequency (PF)
        tic;
        [~, maxIndex] = max(P1);
        pf_features(i, 1:end-1) = f(maxIndex);
        time_pf = toc;
        time_pf_class = time_pf_class + time_pf;

        % 4. Power Spectral Density (PSD)
        tic;
        psd_features(i, 1:end-1) = bandpower(P1, f, 'psd');
        time_psd = toc;
        time_psd_class = time_psd_class + time_psd;

        % 5. Total Power (TP)
        tic;
        tp_features(i, 1:end-1) = sum(P1);
        time_tp = toc;
        time_tp_class = time_tp_class + time_tp;

        % Time-frequency domain features
        for ch = 1:size(emg_filtered, 2)
            % Apply CWT to each channel separately
            [wt, ~] = cwt(curwin(:, ch), 'amor');
            tic;
            % 1. Energy in Wavelet Sub-bands
            tf_energy_features(i, ch) = sum(abs(wt), 'all');
            time_tf_energy = toc;
            time_tf_energy_class = time_tf_energy_class + time_tf_energy;
        end
    end
```

```
    % Display total processing time for each feature type for the current class
    fprintf('Total processing time for MAV features (class %d): %.2f seconds\n',
class_label, time_mav_class);
    fprintf('Total processing time for RMS features (class %d): %.2f seconds\n',
class_label, time_rms_class);
    fprintf('Total processing time for VAR features (class %d): %.2f seconds\n',
class_label, time_var_class);
    fprintf('Total processing time for WL features (class %d): %.2f seconds\n',
class_label, time_wl_class);
    fprintf('Total processing time for ZC features (class %d): %.2f seconds\n',
class_label, time_zc_class);
    fprintf('Total processing time for SSC features (class %d): %.2f seconds\n',
class_label, time_ssc_class);
```

```
    fprintf('Total processing time for WAMP features (class %d): %.2f seconds\n',
class_label, time_wamp_class);
    fprintf('Total processing time for MNF features (class %d): %.2f seconds\n',
class_label, time_mnf_class);
    fprintf('Total processing time for MDF features (class %d): %.2f seconds\n',
class_label, time_mdf_class);
    fprintf('Total processing time for PF features (class %d): %.2f seconds\n',
class_label, time_pf_class);
    fprintf('Total processing time for PSD features (class %d): %.2f seconds\n',
class_label, time_psd_class);
    fprintf('Total processing time for TP features (class %d): %.2f seconds\n',
class_label, time_tp_class);
    %fprintf('Total processing time for VCF features (class %d): %.2f seconds\n',
class_label, time_vcf_class);
    fprintf('Total processing time for TF Energy features (class %d): %.2f seconds\n',
class_label, time_tf_energy_class);


    % Update the total processing time for each feature type across all classes
    total_time_mav = total_time_mav + time_mav_class;
    total_time_rms = total_time_rms + time_rms_class;
    total_time_var = total_time_var + time_var_class;
    total_time_wl = total_time_wl + time_wl_class;
    total_time_zc = total_time_zc + time_zc_class;
    total_time_ssc = total_time_ssc + time_ssc_class;
    total_time_wamp = total_time_wamp + time_wamp_class;
    total_time_mnf = total_time_mnf + time_mnf_class;
    total_time_mdf = total_time_mdf + time_mdf_class;
    total_time_pf = total_time_pf + time_pf_class;
    total_time_psd = total_time_psd + time_psd_class;
    total_time_tp = total_time_tp + time_tp_class;
    %total_time_vcf = total_time_vcf + time_vcf_class;
    total_time_tf_energy = total_time_tf_energy + time_tf_energy_class;
```

```
    % Add the class label as the last column
    mav_features(:, end) = class_label;
    rms_features(:, end) = class_label;
    var_features(:, end) = class_label;
    wl_features(:, end) = class_label;
    zc_features(:, end) = class_label;
    ssc_features(:, end) = class_label;
    wamp_features(:, end) = class_label;
    mnf_features(:, end) = class_label;
    mdf_features(:, end) = class_label;
    pf_features(:, end) = class_label;
    psd_features(:, end) = class_label;
    tp_features(:, end) = class_label;
    %vcf_features(:, end) = class_label;
    tf_energy_features(:, end) = class_label;


    % Step 5: Store features for the current class
```

```
    all_mav_features = [all_mav_features; mav_features];
    all_rms_features = [all_rms_features; rms_features];
    all_var_features = [all_var_features; var_features];
    all_wl_features = [all_wl_features; wl_features];
    all_zc_features = [all_zc_features; zc_features];
    all_ssc_features = [all_ssc_features; ssc_features];
    all_wamp_features = [all_wamp_features; wamp_features];
    all_mnf_features = [all_mnf_features; mnf_features];
    all_mdf_features = [all_mdf_features; mdf_features];
    all_pf_features = [all_pf_features; pf_features];
    all_psd_features = [all_psd_features; psd_features];
    all_tp_features = [all_tp_features; tp_features];
    %all_vcf_features = [all_vcf_features; vcf_features];
    all_tf_energy_features = [all_tf_energy_features; tf_energy_features];


    % Display a message indicating data processing is complete for the current class
    fprintf('Features for class %d processed successfully.\n', class_label);
end
```

```
% Display total processing time for each feature type across all classes
fprintf('Total processing time for MAV features across all classes: %.2f seconds\n',
total_time_mav);
fprintf('Total processing time for RMS features across all classes: %.2f seconds\n',
total_time_rms);
fprintf('Total processing time for VAR features across all classes: %.2f seconds\n',
total_time_var);
fprintf('Total processing time for WL features across all classes: %.2f seconds\n',
total_time_wl);
fprintf('Total processing time for ZC features across all classes: %.2f seconds\n',
total_time_zc);
fprintf('Total processing time for SSC features across all classes: %.2f seconds\n',
total_time_ssc);
fprintf('Total processing time for WAMP features across all classes: %.2f seconds\n',
total_time_wamp);


fprintf('Total processing time for MNF features across all classes: %.2f seconds\n',
total_time_mnf);
fprintf('Total processing time for MDF features across all classes: %.2f seconds\n',
total_time_mdf);
fprintf('Total processing time for PF features across all classes: %.2f seconds\n',
total_time_pf);
fprintf('Total processing time for PSD features across all classes: %.2f seconds\n',
total_time_psd);
fprintf('Total processing time for TP features across all classes: %.2f seconds\n',
total_time_tp);


fprintf('Total processing time for TF Energy features across all classes: %.2f seconds\n',
total_time_tf_energy);
```

```
% Save each combined feature file separately as CSV files with headers
% Define the header
header = {'Feature1', 'Feature2', 'Feature3', 'Feature4', 'Feature5', ...
    'Feature6', 'Feature7', 'Feature8', 'Feature9', 'Feature10', 'Class'};
% Function to write matrix with headers to CSV
write_features_with_headers = @(filename, data, header) writetable(array2table(data,
'VariableNames', header), filename);
% Write each feature set with headers
write_features_with_headers(fullfile(save_path, 'all_mav_features.csv'), all_mav_features,
header);
write_features_with_headers(fullfile(save_path, 'all_rms_features.csv'), all_rms_features,
header);
write_features_with_headers(fullfile(save_path, 'all_var_features.csv'), all_var_features,
header);
write_features_with_headers(fullfile(save_path, 'all_wl_features.csv'), all_wl_features,
header);
write_features_with_headers(fullfile(save_path, 'all_zc_features.csv'), all_zc_features,
header);
write_features_with_headers(fullfile(save_path, 'all_ssc_features.csv'), all_ssc_features,
header);
write_features_with_headers(fullfile(save_path, 'all_wamp_features.csv'),
all_wamp_features, header);
write_features_with_headers(fullfile(save_path, 'all_mnf_features.csv'), all_mnf_features,
header);
write_features_with_headers(fullfile(save_path, 'all_mdf_features.csv'), all_mdf_features,
header);
write_features_with_headers(fullfile(save_path, 'all_pf_features.csv'), all_pf_features,
header);
write_features_with_headers(fullfile(save_path, 'all_psd_features.csv'), all_psd_features,
header);
write_features_with_headers(fullfile(save_path, 'all_tp_features.csv'), all_tp_features,
header);
write_features_with_headers(fullfile(save_path, 'all_tf_energy_features.csv'),
all_tf_energy_features, header);
```

```
% 1. Combine all time-domain features
Combined_Time_Domain_Feature_without_label = [all_mav_features(:, 1:end-1),
all_rms_features(:, 1:end-1), all_var_features(:, 1:end-1), ...
                                all_wl_features(:, 1:end-1), all_zc_features(:, 1:end-1),
all_ssc_features(:, 1:end-1), ...
                                all_wamp_features(:, 1:end-1)];
Combined_Time_Domain_Feature = [Combined_Time_Domain_Feature_without_label,
all_mav_features(:, end)];  % Append the class labels


% 2. Combine all frequency-domain features
Combined_Frequency_Domain_Feature_without_label = [all_mnf_features(:, 1:end-1),
all_mdf_features(:, 1:end-1), all_pf_features(:, 1:end-1), ...
                                    all_psd_features(:, 1:end-1), all_tp_features(:,
1:end-1)];
Combined_Frequency_Domain_Feature = [Combined_Frequency_Domain_Feature_without_label,
all_mnf_features(:, end)];  % Append the class labels
% 3.Time-frequency domain features
```

```
Time_Frequency_Domain_Feature_without_label = [all_tf_energy_features(:, 1:end-1)];
Time_Frequency_Domain_Feature = [Time_Frequency_Domain_Feature_without_label,
all_tf_energy_features(:, end)];  % Append the class labels
% 4. Combine all time,frequency and Time-Frequency domain features
Combined_Time_Freq_and_Time_Freq_Domain_Feature_without_label =
[Combined_Time_Domain_Feature_without_label,Combined_Frequency_Domain_Feature_without_labe
l,Time_Frequency_Domain_Feature_without_label];
Combined_Time_Freq_and_Time_Freq_Domain_Feature =
[Combined_Time_Freq_and_Time_Freq_Domain_Feature_without_label, all_mnf_features(:, end)
];  % Append the class labels


% Define the header dynamically based on the number of features
num_time_domain_features = size(Combined_Time_Domain_Feature, 2) - 1;  % Excluding the
class label
num_frequency_domain_features = size(Combined_Frequency_Domain_Feature, 2) - 1;  %
Excluding the class label
num_time_frequency_domain_features = size(Time_Frequency_Domain_Feature, 2) - 1;  %
Excluding the class label
num_time_freq_and_time_freq_domain_features =
size(Combined_Time_Freq_and_Time_Freq_Domain_Feature, 2) - 1;  % Excluding the class label


header_combined_time = arrayfun(@(x) sprintf('Feature%d', x), 1:num_time_domain_features,
'UniformOutput', false);
header_combined_freq = arrayfun(@(x) sprintf('Feature%d', x),
1:num_frequency_domain_features, 'UniformOutput', false);
header_time_freq = arrayfun(@(x) sprintf('Feature%d', x),
1:num_time_frequency_domain_features, 'UniformOutput', false);
header_combined_time_freq_and_time_freq= arrayfun(@(x) sprintf('Feature%d', x),
1:num_time_freq_and_time_freq_domain_features, 'UniformOutput', false);
header_combined_time{end+1} = 'Class';
header_combined_freq{end+1} = 'Class';
header_time_freq{end+1} = 'Class';
header_combined_time_freq_and_time_freq{end+1} = 'Class';
% Function to write matrix with headers to CSV
write_features_with_headers = @(filename, data, header) writetable(array2table(data,
'VariableNames', header), filename);
% Save combined feature files
write_features_with_headers(fullfile(save_path, 'Combined_Time_Domain_Feature.csv'),
Combined_Time_Domain_Feature, header_combined_time);
write_features_with_headers(fullfile(save_path, 'Combined_Frequency_Domain_Feature.csv'),
Combined_Frequency_Domain_Feature, header_combined_freq);
%write_features_with_headers(fullfile(save_path,
'Time_Frequency_Domain_Feature.csv'),Time_Frequency_Domain_Feature, header_time_freq );
write_features_with_headers(fullfile(save_path, 'Combined_Time_Freq_and_Time-Freq
Domain_Feature.csv'), Combined_Time_Freq_and_Time_Freq_Domain_Feature,
header_combined_time_freq_and_time_freq);
% Display completion message
disp('Combined feature sets saved successfully as CSV files.');

% End overall timing
endTime = datetime('now');
elapsedTime = endTime - startTime;
fprintf('Total execution time: %s\n', string(elapsedTime));
```

2: Python Code for Hybrid model.

```
import pandas as pd  # For data handling
import numpy as np  # For numerical operations
from sklearn.model_selection import train_test_split  # For splitting the data
from sklearn.preprocessing import StandardScaler  # For scaling the data
from sklearn.neighbors import KNeighborsClassifier  # For the k-NN model
from sklearn.metrics import accuracy_score, classification_report  # For model evaluation
import matplotlib.pyplot as plt  # For plotting

# List of data files with paths
files = {


   'combined_Hybrid':r'C:\Users\c3054367\OneDrive - Newcastle University\lab\Final_Hybrid model_1008\Hybrid_MATLAB_Output\Combined_Hybrid.csv',


}



# Function to train and evaluate a k-NN classifier on a given dataset
def train_and_evaluate(file, label='Class'):
   try:
      # Step 1: Load the dataset
      data = pd.read_csv(file)

      # Step 2: Split the dataset into features and target variable
      X = data.drop(label, axis=1)
      Y = data[label]

      # Step 3: Split the dataset into training and testing sets
      X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
```

```
# Step 4: Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 5: Train the k-NN Classifier
classifier = KNeighborsClassifier(n_neighbors=5)  # You can adjust n_neighbors
classifier.fit(X_train, Y_train)

# Step 6: Predict the labels on the test set
Y_pred = classifier.predict(X_test)

# Step 7: Evaluate the classifier accuracy for testing data
accuracy_test = accuracy_score(Y_test, Y_pred)
print(f'Accuracy for testing on {file}: {accuracy_test:.2%}')

# Step 8: Predict the labels on the training set
Y_train_pred = classifier.predict(X_train)

# Evaluate the classifier accuracy for training data
accuracy_train = accuracy_score(Y_train, Y_train_pred)
print(f'Accuracy for training on {file}: {accuracy_train:.2%}')

# Plot the accuracy (optional)
plt.figure(figsize=(5, 4))  # Adjust figure size as desired
plt.bar(['Training', 'Testing'], [accuracy_train, accuracy_test], color=['skyblue', 'lightgreen'])
plt.xlabel('Data Split')
plt.ylabel('Accuracy')
plt.title(f'Training vs Testing Accuracy for {file}')
plt.grid(axis='y')
plt.tight_layout()  # Adjust layout for better readability
plt.show()
```

```
        # Print classification report
        print('Classification Report:')
        print(classification_report(Y_test, Y_pred))

    except FileNotFoundError:
        print(f"File not found: {file}")
    except Exception as e:
        print(f"An error occurred while processing {file}: {e}")

# Train and evaluate a classifier for each dataset
for label, file in files.items():
    train_and_evaluate(file)
```